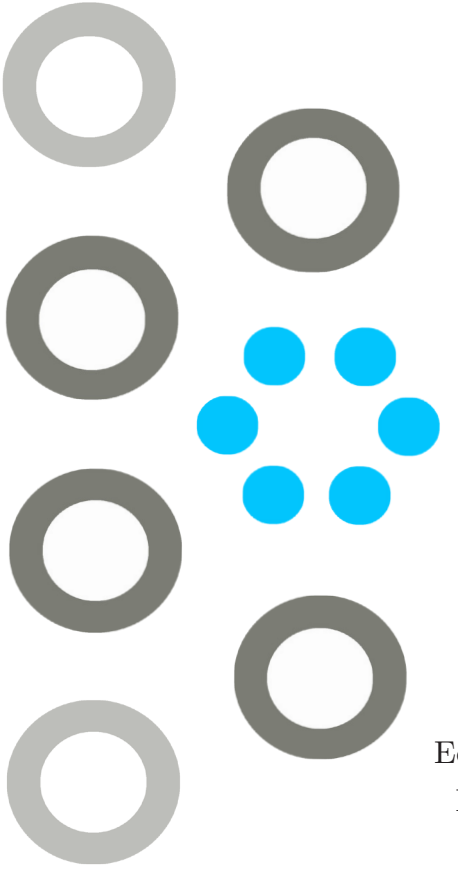
UNIVERSIDAD DE CUENCA
Facultad de Ingeniería
Carrera de Ingeniería de Sistemas



Micro IoT

Metodología

para la creación de aplicaciones basadas en Microservicios para soluciones de Internet de las Cosas en Ambientes de Vida Asistidos

Edwin Fernando Cabrera Alvarado - 0105777122
Paola Johana Cárdenas Cárdenas - 0107462756
Autores

Ing. Irene Priscila Cedillo Orellana, PhD. - 0102815842
Directora

Trabajo de titulación previo a la obtención del título de Ingeniero de Sistemas

Cuenca-Ecuador 2018

Resumen

Los microservicios junto con tecnologías como Internet de las Cosas (*Internet of Things - IoT*) , Cloud Computing, entre otras, han revolucionado el modelo de negocio de compañías orientadas a proporcionar servicios de asistencia, monitoreo y vigilancia en entornos de vida asistidos (*Ambient Assisted Living – AAL*); sin embargo, el desarrollo de aplicaciones basado en microservicios, rompe el esquema de desarrollo de software tradicional y aún más, el esquema organizacional, siendo una arquitectura que cada vez tiene mayor aceptación.

Por otro lado, las metodologías ágiles de desarrollo de software, resaltan por su sencillez, tanto en su aprendizaje como en su aplicación, permitiendo a quienes las emplean la obtención de un producto de software confiable, de calidad y con un diseño que hace frente a los cambios continuos con entregas tempranas; con este estudio se motiva la adopción de metodologías de desarrollo de software ágiles con arquitecturas de microservicios y se presenta una instancia de la misma para soluciones de software en entornos de IoT para AAL.

Consecuentemente, este trabajo de titulación presenta MicroIoT, una metodología para la definición, creación y despliegue de microservicios basada en metodologías ágiles, para soluciones de IoT desplegadas en AAL, la cual;alineada con el enfoque organizacional DevOps, el cual abarca adecuadamente el desarrollo y las operaciones sobre microservicios. Además, se ha diseñado e implementado un sistema de software de IoT para AAL, como una instancia desarrollada con la metodología propuesta. MicroIoT ha sido evaluada empíricamente mediante un cuasi-experimento realizado con profesionales y estudiantes de la carrera de Ingeniería de Sistemas de la Universidad de Cuenca.

Palabras Clave: Metodología, MicroIoT, Arquitectura de Microservicios, Internet de las Cosas, Ambientes de Vida Asistidos.

Abstract

Microservices along with technologies such as Internet of Things (IoT), Cloud Computing, among others, have revolutionized the business model of companies oriented to provide assistance, monitoring and surveillance services in Ambient Assisted Living (AAL); However, the development of applications based on microservices, breaks the traditional software development scheme and even more, the organizational scheme, being an architecture that is increasingly accepted.

On the other hand, the agile methodologies of software development stand out for their simplicity, both in their learning and in their application, allowing those who use them to obtain a reliable software product of quality and with a design that faces continuous changes with early deliveries; with this study the adoption of agile software development methodologies with microservices architectures is motivated and an instance of it is presented for software solutions in IoT environments for AAL.

Consequently, this study presents MicroIoT, a methodology for the definition, creation and deployment of microservices based on agile methodologies, for IoT solutions deployed in AAL, which is aligned with the DevOps organizational approach, which adequately covers the development and operations on microservices. In addition, an IoT software system for AAL has been designed and implemented, as an instance developed with the proposed methodology. MicroIoT has been evaluated empirically through a quasi-experiment carried out with professionals and students of the Systems Engineering Career in the University of Cuenca.

Keywords: Methodology, MicroIoT, Microservice Architecture, Internet of Things, Ambient Assisted Living.

Índice general

Índice de Figuras

Índice de Tablas

Cláusula de Propiedad Intelectual

Edwin Fernando Cabrera Alvarado, autor del trabajo de titulación "Metodología para la creación de aplicaciones basadas en Microservicios para soluciones de Internet de las Cosas en Ambientes de Vida Asistidos", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, 14 de Septiembre de 2018.

Edwin Fernando Cabrera Alvarado

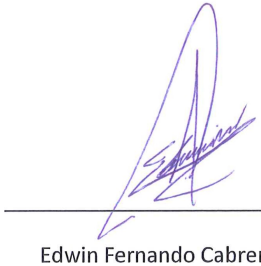
C.I: 0105777122

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Edwin Fernando Cabrera Alvarado en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "Metodología para la creación de aplicaciones basadas en Microservicios para soluciones de Internet de las Cosas en Ambientes de Vida Asistidos", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 14 de Septiembre de 2018.



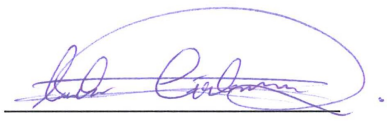
Edwin Fernando Cabrera Alvarado

C.I: 0105777122

Cláusula de Propiedad Intelectual

Paola Johana Cárdenas Cárdenas, autora del trabajo de titulación "Metodología para la creación de aplicaciones basadas en Microservicios para soluciones de Internet de las Cosas en Ambientes de Vida Asistidos", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, 14 de Septiembre de 2018.



Paola Johana Cárdenas Cárdenas

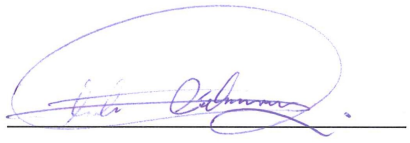
C.I: 0107462756

Cláusula de licencia y autorización para publicación en el Repositorio Institucional

Paola Johana Cárdenas Cárdenas en calidad de autor y titular de los derechos morales y patrimoniales del trabajo de titulación "Metodología para la creación de aplicaciones basadas en Microservicios para soluciones de Internet de las Cosas en Ambientes de Vida Asistidos", de conformidad con el Art. 114 del CÓDIGO ORGÁNICO DE LA ECONOMÍA SOCIAL DE LOS CONOCIMIENTOS, CREATIVIDAD E INNOVACIÓN reconozco a favor de la Universidad de Cuenca una licencia gratuita, intransferible y no exclusiva para el uso no comercial de la obra, con fines estrictamente académicos.

Asimismo, autorizo a la Universidad de Cuenca para que realice la publicación de este trabajo de titulación en el repositorio institucional, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación Superior.

Cuenca, 14 de Septiembre de 2018.



Paola Johana Cárdenas Cárdenas

C.I: 0107462756

Dedicatoria

A mis padres Joel y Silvia, que gracias a Dios les permitieron brindarme todo su cariño y esfuerzo, y que supieron inculcarme los cimientos para la formación de mi vida personal y profesional.

A mi hermana Karina, quien fue mi inspiración para seguir esta carrera.

A mis tíos, primos y abuelitos, quienes han sido un apoyo permanente en el transcurso de mi carrera universitaria.

A Paola, mi amiga y compañera de tesis, que gracias a su responsabilidad y esfuerzo logramos concluir exitosamente este trabajo.

Edwin Cabrera A.

Dedicatoria

A mis padres, Angel y Elsa, gracias a su esfuerzo y sus enseñanzas que han contribuido en mi proceso de formación como persona y como profesional para lograr cumplir con este objetivo, le doy gracias a Dios por darme padres tan maravillosos y a mi familia.

A William, Isabel y Steven en quienes siempre encontre sentimientos de apoyo y motivación. A todas aquellas personas especiales en mi vida, aquellas que siempre me dieron una palabra de aliento y celebran conmigo cada meta alcanzada.

Por último a mi compañero de tesis Edwin, gracias por ser un excelente amigo y compañero, juntos hemos realizado un trabajo que nos enorgullece y plasma lo aprendido en esta maravillosa carrera.

Paola Cárdenas C.

Agradecimientos

A nuestra directora, Ing. Priscila Cedillo, una gran profesional, quien no sólo nos ha transmitido sus amplios conocimientos, sino también nos ha dado las pautas y su ayuda incondicional para la consecución del presente trabajo.

A nuestras familias por el cariño y todo el apoyo brindado durante nuestra vida universitaria.

A Cristina Peralta, quien ha sido más que una gran amiga le agradecemos por su colaboración. Agradecemos también a nuestros amigos y compañeros, con quienes compartimos largos años de estudio; y a todas las personas, que directa o indirectamente estuvieron con nosotros en la realización y culminación con éxito del presente trabajo.

A todos ustedes mil gracias.

Edwin Cabrera A. **Paola Cárdenas C.**

Capítulo 1

Introducción

Este capítulo tiene como objetivo presentar la motivación, contexto, problemática actual y la solución propuesta en el presente trabajo de titulación, incluyendo las hipótesis, objetivos generales y específicos. Adicionalmente, se muestran los aportes científicos generados con el desarrollo de este trabajo.

1.1. Motivación y Contexto

Actualmente, en la mayoría de actividades, se encuentra presente la tecnología, mediante diferentes tipos de dispositivos que hacen uso del paradigma de Internet de las Cosas (IoT - *Internet of Things*), el cual se define como la infraestructura mundial para la sociedad de la información que propicia la prestación de servicios avanzados mediante la interconexión de objetos físicos y virtuales gracias a la interoperabilidad de tecnologías de la información y las redes de comunicación presentes y futuras (Benítez-Gutiérrez, 2017).

El primer modelo de la arquitectura del Internet de las cosas se compone de tres capas principales las cuales son: capa de percepción, capa de red y capa de aplicación (Sethi and Sarangi, 2017). IoT junto con tecnologías como microservicios, cloud computing y otras, han revolucionado el modelo de negocio de compañías orientadas a proporcionar un servicio más personalizado en base a gustos o necesidades de sus clientes fomentando la búsqueda de vincular este tipo de tecnologías (Butzin et al., 2016).

Los microservicios son aplicados en IoT, debido a características como: interoperabilidad, resiliencia, escalabilidad, mantenibilidad y otras caracteris-

ticas que una arquitectura de microservicios provee, como lo recalca Krylovskiy et al. (2015), quien desarrolló una plataforma IoT para *Smart Cities* aplicando una arquitectura de microservicios; por otro lado, Castro et al. (2017), menciona que se han desarrollado diferentes aplicaciones de IoT orientadas a Ambientes de Vida Asistidos (AAL - *Ambient Assited Living*), tales como: cuidado personalizado, mejora de la relación paciente-médico (a través de dispositivos que pueden ser accesibles para los pacientes), sistema de asistencia médica, entre otras, teniendo un buen nivel de aceptación como lo señalan Duplaga (2013) y Yang et al. (2014).

En la actualidad, las personas tienen un cúmulo de servicios tecnológicos a su disposición, sin embargo vivimos en una era de fuerte transición en donde aún hay mucho por mejorar. Teniendo en cuenta las investigaciones de salud, el número de adultos mayores que necesitan más apoyo para realizar actividades cotidianas en su vida está creciendo rápidamente (Hail and Fischer, 2015), del mismo modo la mejora de la calidad de vida de las personas con necesidades especiales ha adquirido una creciente atención y se han propuesto soluciones que permiten aprovechar los conocimientos y la contribución al desarrollo que pueden hacer las personas con discapacidades. Todo esto, junto a la lucha constante de gobiernos, organizaciones y personas, para lograr la participación, equidad e inclusión, ha dado lugar a un paradigma que busca la defensa de la diversidad y de soluciones innovadoras para el futuro de las personas (de Lorenzo García, 2012).

En otro orden, dentro de la Universidad de Cuenca se ha conformado un grupo de investigación, dedicado a la definición del uso e integración de tecnologías que soportan ambientes de vida asistidos, con la finalidad de establecer lineamientos para la producción de sistemas, que no solo estén enfocados a la funcionalidad, como ocurre muchas de las veces con los sistemas que se orientan a especificaciones técnicas y funcionales, sin tener una visión a futuro del software ni de su evolución, olvidando así la importancia que tiene la Ingeniería de Software al momento de desarrollar una solución mantenible, interoperable y escalable.

1.2. Planteamiento del Problema

Existe un problema en el desarrollo de sistemas de asistencia, especialmente con el uso de las nuevas tecnologías, las cuales tienen una gran proyección de uso en el futuro, además, como Evans (2011) menciona, para el año 2020 existirán 50 billones de dispositivos conectados a Internet. Con independencia del atractivo o la incomprensión del concepto de IoT, el temor a quedar relegado por los competidores o la presión por hacer algo novedoso hace que

las empresas a menudo se precipiten hacia proyectos de IoT sin definir claramente los objetivos, la propuestas de valor o la idoneidad de las herramientas; generando así un alto índice de fracaso de los proyectos de IoT, y un enorme desengaño entre los clientes, debido a falta de validaciones del cliente y falta de planificación para el desarrollo del proyecto Gubbi et al. (2013).

Por otro lado, se considera la inclusión y adaptación de lugares para personas con necesidades especiales (discapacitados, adultos mayores, etc.) como un problema latente en la sociedad. *Ambient Assisted Living* (AAL), incluye sistemas técnicos para ayudar a personas con necesidades especiales en su vida cotidiana, su objetivo es mantener y promover la independencia de las personas con necesidades especiales, así como mejorar la calidad de los servicios de asistencia y apoyo Georgieff (2008). El uso de IoT junto con AAL, pretende solucionar muchos de los problemas de asistencia, inclusión e implementación de ambientes asistidos para personas con necesidades especiales.

De ahí, se requieren aplicaciones que funcionen de manera distribuida y que permitan interconectar soluciones tecnológicas, recuperando información de interés, monitoreando el estado y las necesidades de las personas, entre otras. La distribución de las aplicaciones, puede verse solventada, a través de artefactos de software publicados como servicios, los mismos que pueden ser construidos y distribuidos, atendiendo diversos criterios, que pueden ir desde la construcción por funcionalidad, por características no funcionales, en casos concretos según el hardware al que soportan, etc. Por tanto, éstos paradigmas han desplazado a los sistemas monolíticos y se empieza a crear software visto como una convergencia de servicios distribuidos y aplicaciones que dirijan a cada uno de ellos de manera aislada pero que a su vez permitan unir su funcionalidad hacia un objetivo común, la asistencia al usuario con discapacidad o limitaciones física o cognitivas (de Lorenzo García, 2012).

1.3. Solución Propuesta

Consecuentemente, la contribución de este trabajo de titulación es aplicar la Ingeniería de Software para generar una metodología que facilite el proceso de desarrollo de sistemas y aplicaciones para ambientes de vida asistidos, mediante la creación de microservicios para soportar a dispositivos de IoT. Para lograr esto, se requiere realizar un estudio del arte para determinar las técnicas aplicadas en *Ambient Assisted Living* (AAL), determinar los diferentes requerimientos para cada uno de los tipos de dispositivos que pueden conformar el ecosistema de IoT en dichos ambientes y que sirvan de apoyo para una mejor calidad de vida de personas con discapacidades o adultos mayores. La metodología se alinea a diversos paradigmas de desarrollo que representan nuevos

métodos y formas de desarrollo y que mejoran sustancialmente el software que se ha producido hasta el momento.

1.4. Hipótesis y Objetivos

Se definen a continuación las hipótesis y los objetivos.

1.4.1. Hipótesis propuestas

H0: La metodología propuesta, no se percibe como una solución útil para el desarrollo de aplicaciones basadas en microservicios, en soluciones de Internet de las Cosas para Ambientes de Vida Asistidos.

Ha: La metodología propuesta, resulta de gran utilidad en el desarrollo de aplicaciones basadas en microservicios para soluciones de Internet de las Cosas en Ambientes de Vida Asistidos.

1.4.2. Objetivo General

Proponer una metodología para la creación de aplicaciones basadas en microservicios para Ambientes de Vida Asistidos (AAL - *Ambient Assisted Living*) que integre Internet de las Cosas (IoT) y sea una herramienta para implementar soluciones que aporten favorablemente a la calidad de vida de personas con discapacidad y adultos mayores.

1.4.3. Objetivos específicos

- Llevar a cabo un estudio del arte acerca de las técnicas y estrategias actualmente utilizadas para el desarrollo e implementación de Ambientes de Vida Asistidos.
- Identificar los requerimientos generales de aplicaciones basadas en Microservicios para soluciones de Internet de las Cosas en Ambientes de Vida Asistidos, establecer aquellos importantes e imprescindibles, incluyendo aspectos funcionales y no funcionales.
- Elaborar una metodología para la planificación de aplicaciones para ambientes asistidos dirigidos a personas pertenecientes a sectores vulnerables de la sociedad.
- Identificar tecnologías adecuadas para la construcción de la metodología, evaluarlas, compararlas y recomendarlas según los escenarios propuestos.

- Evaluar la metodología desarrollada en un escenario puntual y real, con un estudio de caso o prueba de conceptos.

1.5. Metodología de la investigación

Para la elaboración de la propuesta, se ha seguido una metodología estructurada conforme al modelo de transferencia tecnológica de Gorschek et al. (2006), el cual contempla ocho actividades que pretenden encontrar una solución realista mediante un proceso iterativo de validación empírica de soluciones candidatas, como se muestra en la Figura 1.1 y se detallan a continuación.

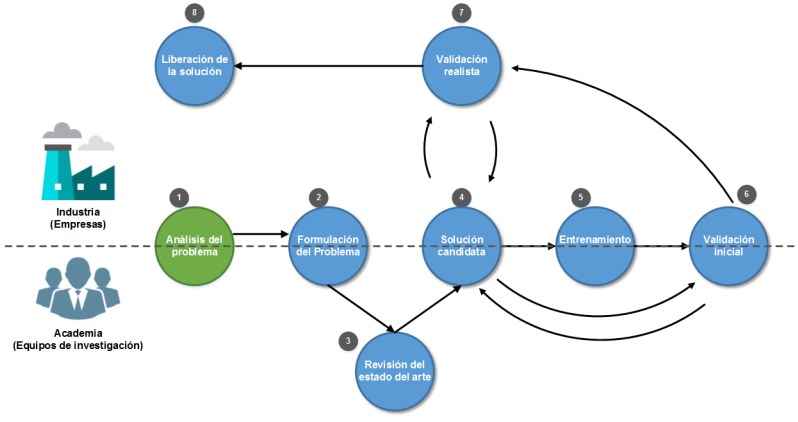


Figura 1.1: Metodología de la investigación (Fuente: Elaboración propia).

1. Análisis del problema: consiste en comprender el problema que da origen y propósito a la investigación, para lo cual se observa el dominio, en este caso Internet de las Cosas (IoT - *Internet of Things*), específicamente entornos de Ambientes de Vida Asistidos (AAL - *Ambient Assisted Living*), y se identifican las necesidades de la industria, en empresas que ofrecen servicios de IoT para AAL, o las organizaciones que se dedican a la creación de aplicaciones y sistemas de software IoT para AAL.

2. Formulación del problema: luego de analizar el problema, se lo formula de forma clara y precisa, incluyendo factores de contexto, objetivos que la investigación persigue, planteamiento de preguntas de investigación y justificación del estudio.

3. Revisión del estado del arte: por medio de una revisión sistemática de

literatura se analiza el estado actual de la investigación, a través de la revisión de estudios primarios existentes para determinar las brechas existentes con respecto al tema. Para ello se siguen los lineamientos que propone Kitchenham et al. (2009), los cuales tienen como objetivo reflejar los problemas o necesidades específicas en la investigación de Ingeniería de Software, a través de tres fases: (i) planificación, (ii) conducción o ejecución de la revisión y (iii) difusión de resultados.

4. Solución candidata: se propone una metodología preliminar para dar solución al problema establecido, la metodología debe tener en bases sólidas, reforzada con procesos existentes que han sido probados o son usados por grandes empresas, el aporte se evidencia en la creación de una metodología para la creación de sistemas de software centrada en la Arquitectura de Microservicios en un entorno de Internet de las Cosas para Ambientes de Vida Asistidos, la cual no ha sido encontrada en la revisión de la literatura.

5. Entrenamiento: consiste en una actividad de tipo incremental, en donde se busca proporcionar el conocimiento necesario a los profesionales del área para generar una vista general de la solución propuesta. Dicha solución será más tarde depurada a través de las distintas actividades de refinamiento (actividades 5 y 6).

6. Validación inicial: la validación de la solución propuesta se la realiza en un entorno de laboratorio, para lo cual se utiliza un caso de estudio basado en cuasi-experimentos con alumnos y profesionales de la carrera de Ingeniería de Sistemas.

7. Validación realista: se aplica un entorno real industrial para llevar a cabo casos de estudio o experimentos controlados.

8. Liberación de la solución: Se hace una valoración de los resultados obtenidos y se preparan las herramientas y material requerido para su uso.

En este trabajo se han cubierto las primeras seis actividades de este modelo, proponiendo como trabajos futuros las actividades de validación realista y liberación de la solución.

1.6. Estructura del trabajo

En este capítulo se ha presentado la motivación, el planteamiento del problema, las hipótesis que serán contrastadas, los objetivos del trabajo de titulación, y la metodología que será empleada durante el proceso de investigación.

El proceso de investigación seguido en este trabajo de titulación se describe en varios capítulos que se corresponden con las actividades del modelo de transferencia de tecnología propuesto por Gorschek et al. (2006). A continuación se describen brevemente los capítulos presentados a lo largo de este documento.

- Capítulo 2: Marco teórico

  Se presenta un compendio de descripciones y nociones clave sobre conceptos teóricos, revisión sistemática, Ingeniería de Requerimientos, arquitecturas de desarrollo de software, Internet de las Cosas, entre otros.

- Capítulo 3: Estado del arte

  Se muestra el proceso y los resultados de la aplicación de la metodología de Kitchenham et al. (2009), para esclarecer el estado del arte, por medio de una revisión sistemática de literatura, centrada en "gestión de servicios en soluciones".

- Capítulo 4: Metodología propuesta.

  Se describe a detalle la metodología propuesta, mostrando, sus fases, tareas, roles involucrados en cada una de éstas, entradas y salidas de cada proceso.

- Capítulo 5: Implementación

  Se presenta la implementación de una aplicación, haciendo uso del método propuesto en el presente trabajo.

- Capítulo 6: Validación mediante caso de uso

  Se realiza la validación de la implementación del método propuesto en el capítulo 4, para lo cual se utiliza la metodología de Runeson and Höst (2009), aplicada a la implementación realizada en el capítulo 5.

- Capítulo 7: Conclusiones y trabajos futuros

  Se presenta la información más relevante obtenida con el desarrollo de este trabajo. Adicionalmente se proponen trabajos futuros relacionados al presente.

- Glosario

  Contiene una descripción de términos empleados en el documento.

- Apéndices

  Contiene toda la documentación adjunta y generada en el transcurso del presente trabajo de titulación, como son: diagramas y otros documentos

generados durante la implementación del método en el caso de estudio y la validación del mismo.

La Figura 1.2 muestra la estructura del trabajo de titulación, donde se puede observar las distintas fases, flujo de actividades, y relación con los capítulos.

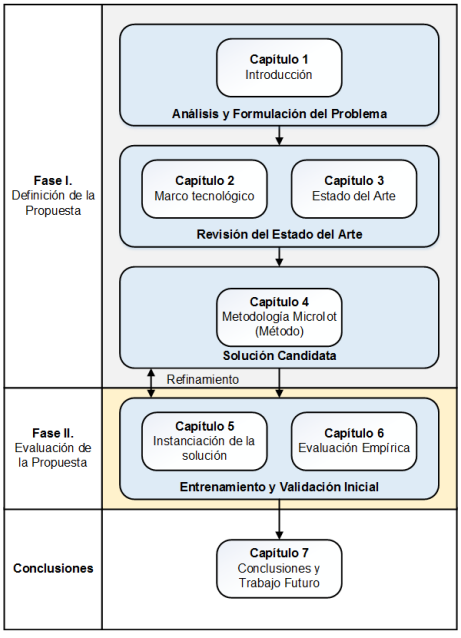


Figura 1.2: Estructura del trabajo de titulación (Fuente: Elaboración propia).

Es importante recalcar que las actividades de validación realista y liberación de la solución de la metodología seguida para el desarrollo de esta propuesta, se vislumbran como trabajos futuros cuando se tenga acceso a validaciones y experimentación empírica en organizaciones que se dediquen a la creación de sistemas de software de Internet de las Cosas. En este sentido, se ha considerado conveniente ejecutar la validación con sujetos experimentales que posean conocimientos sólidos en microservicios y con algún tipo de experiencia en el diseño de sistemas distribuidos u organizacionales.

Edwin Cabrera Alvarado
Paola Cárdenas Cárdenas

Capítulo 2

Marco tecnológico

En este capítulo se conceptualizan las tecnologías que representan los pilares fundamentales para el entendimiento del presente trabajo de titulación, el capítulo está estructurado como sigue: (i) Arquitecturas de Desarrollo de Software, (ii) Internet de las Cosas, (iii) Metodologías Ágiles, (iv) DevOps, (v) Ingeniería de Requerimientos, (vi) Diseño dirigido por el Dominio, (vii) Diseño Dirigido por Capacidades de Negocio, además se describen algunas herramientas para el manejo de microservicios y la automatización de DevOps.

2.1. Arquitecturas de Desarrollo de Software

Una arquitectura de software puede ser considerada como una disciplina de gran importancia dentro de la Ingeniería de Software. Según Fernández (2006), una arquitectura adecuada es pieza clave para alcanzar la satisfacción tanto los requerimientos funcionales como no funcionales de un sistema, mientras que una arquitectura no adecuada puede ser catastrófica dentro de una aplicación o sistema de software complejo. En esta sección se presenta una introducción teórica a las arquitecturas de software más usadas para la construcción de aplicaciones. Algunos antecedentes sobre la forma en que las arquitecturas de software han evolucionado a lo largo de los años, desde los servidores centralizados, a través de una arquitectura cliente-servidor a la Arquitectura Orientada al Servicio (SOA). SOA con el tiempo evolucionó hasta convertirse en Microservicios (Sharma, 2017).

2.1.1. Arquitectura Monolítica

Una arquitectura monolítica describe una aplicación en la que toda la funcionalidad del sistema (ej. acceso a datos, interfaz de usuario, lógica, etcétera) está implementada y mezclada en una sola capa, presentando grandes limitaciones de escalabilidad, mantenibilidad y eficiencia (Gutiérrez, 2010).

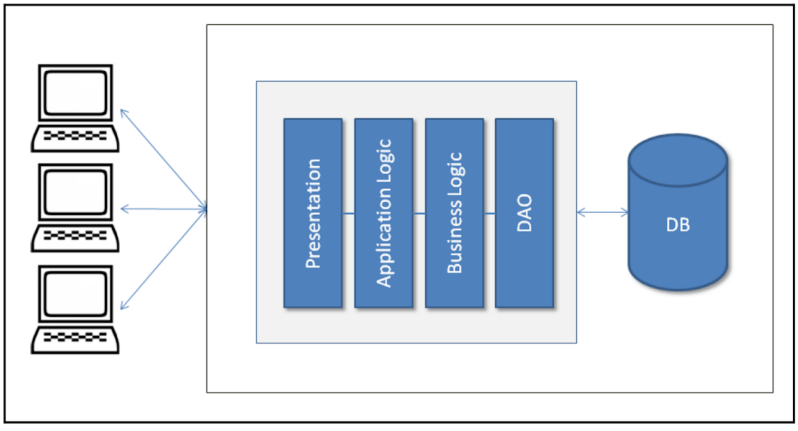


Figura 2.1: Arquitectura monolítica tradicional (Sharma, 2017).

"En los diseños monolíticos tradicionales, todo lo que se incluye en el mismo archivo comprimido como el código de presentación, lógica de la aplicación y el código de lógica de negocio, y DAO y el código relacionado que interactúa con los archivos de bases de datos u otra fuente" (Sharma, 2017).

2.1.2. Arquitectura Orientada a Servicios (SOA)

Como describe Shadija et al. (2017), los servicios son el resultado del encapsulamiento de un número de componentes de software y presentando una interfaz única para ofrecer una función única de negocio.

Según Sharma (2017) SOA es un sistema monolítico con servicios, que surgió cuando las aplicaciones comenzaron a ser desarrolladas en base a servicios, en donde cada componente proporciona cierta funcionalidad a otros componentes o entidades externas. El diagrama de la Figura 2.2 representa la aplicación monolítica con diferentes servicios; aquí los servicios están siendo utilizados con un componente de presentación. Todos los servicios, el componente de presentación, o cualquier otro componente se agrupan.

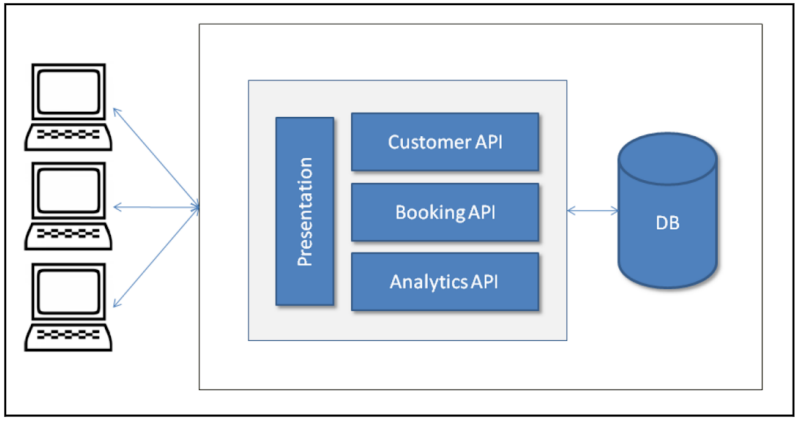


Figura 2.2: Arquitectura Orientada a Servicios (Sharma, 2017).

En el tipo de arquitectura SOA la ejecución de un componente de un servicio puede requerir un componente procedente de otro componente. En la Figura 2.3 se muestran las dependencias entre componentes donde las flechas verdes representan dependencias entre componentes de un mismo servicio, y flechas rojas representan dependencias de un componente de otro servicio. Esta relación de dependencia entre servicios, provoca los se tenga en cuenta ciertos aspectos para el uso de esta arquitectura:

- Al existir una dependencia entre servicios debe existir un mecanismo de gestión de los servicios, por lo que se requiere crear un ESB (*Enterprise Service Bus*) que establece protocolos de comunicación para solucionar este problema.
- Respecto al almacenamiento y gestión de los datos, los servicios requieren establecer un mismo mecanismo de gestión de los datos.
- Los servicios deben ser creados con una misma tecnología.

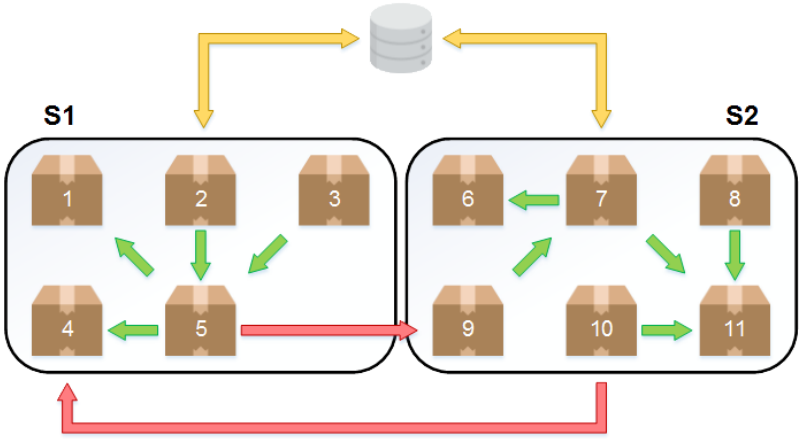


Figura 2.3: Ejemplo arquitectura SOA (Fuente: Elaboración propia).

2.1.3. Arquitectura de Microservicios (MSA)

Un microservicio es definido por Thönes (2015) como: "Una pequeña aplicación que puede implementarse y escalar y puede ser probada de forma independiente y que tiene una responsabilidad única", lo cual implica que no existe una relación de dependencia entre microservicios. Aunque no existe una definición formal para la Arquitectura de Microservicios (MSA) la definición ofrecida por Lewis and Fowler (2014), es la más aceptada, esta definición dice que MSA es un enfoque para desarrollar una sola aplicación como un conjunto de pequeños servicios, cada uno ejecutándose en su propio proceso y comunicándose con mecanismos livianos.

El patrón de Arquitectura de Microservicios, ha sido utilizado por grandes empresas de Internet como Amazon, Netflix y LinkedIn para desplegar grandes aplicaciones en la nube como un conjunto de pequeños servicios que se pueden desarrollar, probar, desplegar, escalar, operar y actualizar de forma independiente, permitiendo que estas empresas logren ganar agilidad, reducir complejidad y escalar sus aplicaciones en la nube de forma más eficiente. Partiendo del ejemplo presentado en la Figura 2.3 se ha modificado para establecer una MSA como se muestra en la Figura 2.4, en la que se observa que los microservicios son independientes entre sí y entre los repositorios de almacenamiento de datos.

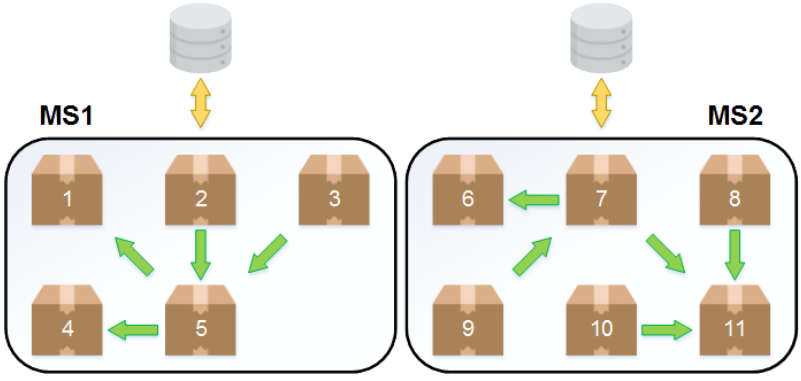


Figura 2.4: Ejemplo aplicación con arquitectura MSA (Fuente: Elaboración propia).

En el enfoque de microservicios, la propiedad de la autocontención es uno de los aspectos centrales. El término autocontención se usa para describir que los servicios deben contener todo lo que necesitan para cumplir su tarea por sí mismos. Esto incluye no sólo su lógica comercial, sino también su *front-end* y *back-end*, así como también las bibliotecas requeridas. Sin embargo, los servicios no deberían ser demasiado grandes para ser mantenibles (Butzin et al., 2016). Cada componente representa la autonomía. Cada componente podría ser desarrollado, construido, probado y desplegado de forma independiente. Aunque el componente de aplicación de interfaz de usuario también podría ser un cliente y consumir los microservicios. En la Figura 2.5 se muestra que la puerta de enlace API (API *Gateway*) proporciona la interfaz donde los diferentes clientes pueden acceder a los servicios individuales.

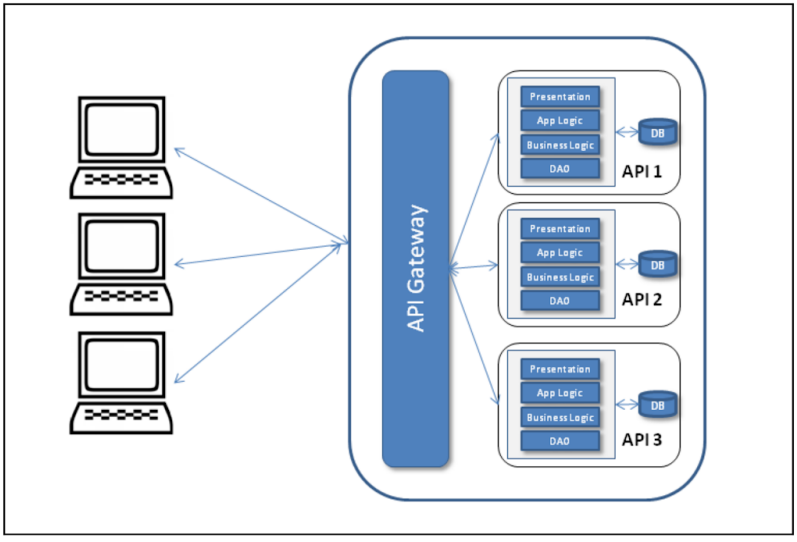


Figura 2.5: Arquitectura de Microservicios (Fuente: (Sharma, 2017)).

A. Patrones de la Arquitectura de Microservicios

La arquitectura de microservicios corresponde al proceso de gestión de los diferentes microservicios a desarrollar; a través de la implementación de un conjunto de patrones, estos patrones presentados por (Richardson, 2014) se agrupan como se muestra en la Figura 2.6.

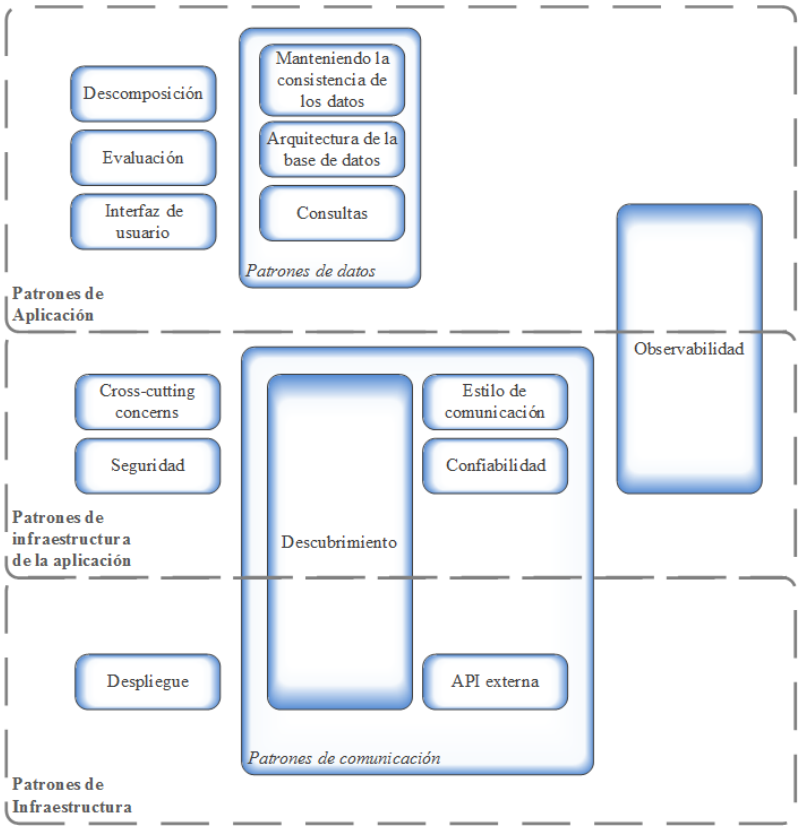


Figura 2.6: Patrones de microservicios (Fuente: Richardson (2014)).

I. **Patrones de descomposición:** Establece el método aplicado para descomponer la aplicación ya sea a través de las capacidades de negocio o aplicando una subdivisión en base a subdominios.

II. **Patrones de evaluación:** Indica patrones que pueden ser implementados para facilitar la automatización de pruebas de servicios propios o consumidos por terceros.

III. **Patrones de interfaz de usuario:** Establece soluciones para implementar una interfaz de usuario de lado del cliente y de lado del servidor.

IV. **Patrones de arquitectura de base de datos:** Establece las consideraciones a tomar respecto a las bases de datos, esto involucra:

- El tipo de arquitectura de base de datos que puede ser una base de datos por servicio, o una base de datos compartida.
- En caso de optar por una base de datos por servicio, se debe considerar una técnica que permita descomponer una consulta para consultar a las diferentes bases de datos.
- En caso de optar por una base de datos, se requiere establecer mecanismos para garantizar la consistencia de los datos a través involucrando sagas (secuencia de transacciones locales), gestión por eventos, *logs*, o disparadores (*triggers*).

V. **Patrones de observabilidad:** Contiene patrones para el monitoreo de las actividades de los usuarios, del estado de los servicios, gestión de las excepciones y *logs*.

VI. **Patrones de preocupaciones transversales:** Las preocupaciones transversales son aquellas preocupaciones que pueden influir en diferentes niveles y perspectivas de la arquitectura tales como métricas que permitan monitorear el rendimiento de la aplicación, gestión de credenciales y servicios externos.

VII. **Patrones de seguridad:** Indica cómo comunicar la identidad de un solicitante a los servicios requeridos.

VIII. **Patrones de estilo de comunicación:** Engloba patrones que facilita la comunicación entre servicios y con los clientes ya sea a través de llamadas a procedimientos remotos, uso de protocolos de mensajería.

IX. **Patrones de confiabilidad:** Proporciona una estrategia para evitar que un servicio se conecte con servicios que se encuentran fallidos.

X. **Patrones de descubrimiento:** Establece como se registra las instancias de los servicios disponibles, así como el proceso de localización de los servicios por lado del cliente y el servidor.

XI. **Patrones de API externa:** Indica el mecanismo de comunicación entre el cliente y los servicios, ya sea a través de un API *Gateway* o mediante un *back-end* como *front-end*.

XII. **Patrones de despliegue:** Muestra la estrategia de despliegue de los microservicios, ya sea a través de *host*, *serverless* (aplicaciones que involucran servicios de terceros), o contenedores.

2.1.4. Diferencias entre SOA y MSA

Tras revisar la revisión conceptual de las arquitecturas SOA y MSA se detallan algunas las ventajas o diferencias más significativas, debido a que muchas veces se confunden los conceptos y tales diferencias no son tan evidentes, debido a que la Arquitectura de Microservicios, como muchos autores lo han descrito es una evolución de la Arquitectura Orientada a Servicios (SOA). Algunas de las ventajas de Microservicios sobre SOA en términos de manejo de servicios de la aplicación y la forma en la que operan en IoT son descritas a continuación:

- Los microservicios ofrecen una granularidad más fina ya que tienen una responsabilidad única.
- No se requiere un *Enterprise Service Bus* (ESB) para establecer un protocolo de comunicación debido a que los microservicios son independientes entre sí.
- Se puede manejar diferentes tipos de mecanismos de almacenamiento de datos dependiendo de la función que se encarga un microservicio.
- SOA proporciona un nivel más alto posible de abstracción de interfaz que MSA; dado que SOA generalmente integra un ESB, que permite la transformación y la mejora de los mensajes de interacción (Rademacher et al., 2017).
- SOA facilita la interacción de servicios, cuyos formatos de mensaje y estructuras difieren. Los sistemas basados en MSA generalmente no comprenden un ESB, la interacción con los servicios sólo es posible si los consumidores usan formatos de mensaje y estructuras directamente respaldadas por los servicios (Rademacher et al., 2017).
- Los ESB también pueden transformar protocolos, por lo que SOA puede admitir una cantidad arbitraria de protocolos de transporte. Esto permite la interacción de los participantes que usan diferentes protocolos. Por el contrario, MSA suele aplicar como máximo dos protocolos diferentes, es decir, uno para la comunicación uno-a-uno y uno para la comunicación de uno-a-muchos escenarios de interacción. Mientras que las interacciones uno a uno dependen principalmente de REST a través de HTTP, las interacciones de uno a muchos siguen un patrón de mensajería de publicación-suscripción (Rademacher et al., 2017).
- En microservicios, se utiliza la capa API en lugar del ESB de SOA. El beneficio de este enfoque API para IoT es que permite cambiar el nivel

de granularidad de los servicios sin afectar a los consumidores. Con el componente API, los consumidores no necesitan cambiar el contrato de servicio o adaptarse ellos mismos (Uviase and Kotonya, 2018).

- El middleware de SOA utiliza un mediador para orquestación de servicios, mejora de mensajes y transformación de protocolos implementada dentro del ESB. En MSA una capa API inteligente para IoT utilizará componentes más pequeños. Estos componentes no están encerrados dentro de la capa API, pero estarán disponibles en tiempo de ejecución para ejecutar funciones similares. Estos componentes compensarán el soporte de protocolos heterogéneos de los que carece de una arquitectura de microservicio estándar (Uviase and Kotonya, 2018).

- Los middleware, tales como el ESB, son muy importantes para SOA, en microservicios es menos complejo, y aunque se puede utilizar un ESB, sólo se utiliza para el transporte de mensajes y no contiene ninguna lógica (Sharma, 2017).

2.1.5. REST (*Representational State Transfer*)

Es un estilo arquitectónico que define un conjunto de restricciones que se utilizarán para crear servicios web, Roy Fielding definió e introdujo el término REST, *Representational State Transfer* en su disertación doctoral, REST es un estilo de arquitectura de software para un sistema hipermedia distribuido como RESTful se refiere a los sistemas que se ajustan a las propiedades, principios y restricciones de la arquitectura REST (Sharma, 2017).

2.2. Internet de las Cosas

Internet de las Cosas (*Internet of Things* - IoT) es una infraestructura global para la sociedad de la información, que permite servicios avanzados interconectando cosas (físicas y virtuales) basadas en tecnologías de información y comunicación interoperables existentes y en evolución. Con IoT, la comunicación se extiende por Internet a todas las cosas que nos rodean (Patel et al., 2016).

IoT cuenta con las siguientes características (Vermesan and Friess, 2014):

- **Interconectividad:** Cualquier cosa puede estar interconectada con la infraestructura global de información y comunicación.

- **Servicios relacionados con las cosas:** IoT es capaz de proporcionar servicios relacionados con las cosas dentro de las limitaciones de las mis-

mas, como la protección de la privacidad y la coherencia semántica entre las cosas físicas y sus elementos virtuales asociados.

- **Heterogeneidad:** Los dispositivos IoT son heterogéneos ya que se basan en diferentes plataformas y redes de hardware. Pueden interactuar con otros dispositivos o plataformas de servicio a través de diferentes redes.

- **Cambios dinámicos:** El estado de los dispositivos cambia dinámicamente, en base a las necesidades o preferencias del usuario, por ejemplo; durmiendo y despertando, conectados y / o desconectados, así como el contexto de los dispositivos, incluida la ubicación y la velocidad. Además, la cantidad de dispositivos puede cambiar dinámicamente.

- **Escala enorme:** El número de dispositivos que deben administrarse y comunicarse entre sí será al menos un orden de magnitud mayor que los dispositivos conectados a la Internet actual, sin embargo; a medida que el entorno IoT aumenta la gestión de datos generados y su interpretación se complica.

Según Vermesan and Friess (2014) entre los principales elementos que componen un dispositivo IoT tenemos:

- **Dispositivos controladores:** Dispositivo físico o programa de software que sirve como punto de conexión entre controladores, sensores, la nube y dispositivos inteligentes.

- **Dispositivos sensores:** Un sensor es un dispositivo que detecta eventos o cambios en su entorno físico y provee una salida correspondiente.

- **Dispositivos actuadores:** Un actuador es un tipo de motor que es responsable de controlar o tomar medidas en un sistema. Este toma una fuente de datos o energía y lo convierte los datos/energía en movimiento para controlar un sistema.

- **Dispositivos inteligentes con sensores, actuadores y controladores integrados:** Artefacto tecnológico que puede conectar el mundo físico con el mundo digital mediante la provisión de capacidades de proyección como monitoreo, detección o accionamiento. Además, viene con capacidad de comunicación para otros sistemas de TI a través de una unidad física externa, o de forma directa.

2.2.1. Un enfoque de Microservicios en Internet de las Cosas.

Diferentes patrones y prácticas del enfoque de microservicio pueden combinarse con las prácticas en IoT. Se cubre la autocontención de los servicios, la supervisión y la prevención de fallas en cascada, coreografía y orquestación, tecnologías de contenedores y el manejo de diferentes versiones de servicios (Butzin et al., 2016).

A. Autocontención

En el enfoque de microservicios, la propiedad de la autocontención es uno de los aspectos centrales, el término autocontención se usa para describir que los servicios deben contener todo lo que necesitan para cumplir su tarea por sí mismos. Adoptar la autocontención en Internet de las Cosas podría generar los siguientes beneficios:

- Al tener el *back-end* como parte del servicio, podemos descuidar las dependencias para el almacenamiento de datos. Los datos guardados por un servicio no deben ser directamente accesibles desde fuera del servicio. Esto impone el uso de la API de servicio y, por lo tanto, desacopla a los consumidores de datos externos de la representación interna de los datos, permitiendo que el modelo de datos interno cambie libremente, al tiempo que se mantiene la interoperabilidad.

- Hacer que cada servicio proporcione su propia interfaz de usuario también permitiría una evolución independiente. Esto también omite la necesidad de un *front-end* centralizado que tenga en cuenta todos los dispositivos posibles que puedan aparecer.

- La limitación para tener la menor cantidad de dependencias posible conduce a un mejor desacoplamiento entre los servicios, un aumento de la autonomía y reduce la cantidad de comunicaciones requeridas en la red en general.

B. Coreografía sobre Orquestación

Cuando hablamos de poner varios servicios juntos, hay múltiples formas de hacerlo. Los conceptos más básicos de hacerlo son la orquestación y la coreografía (Butzin et al., 2016).

ı **Orquestación** significa, una instancia tiene el control de los servicios a los que se llama y lo hace de forma centralizada.

- **Coreografía**, cada participante hace su parte por sí mismo y la aplicación resultante se crea por la suma de los individuos. Las partes individuales llevan a cabo su actividad en función del evento. El inconveniente de implementar coreografía es que no es posible rastrear si todas las acciones requeridas se realizan con éxito. Para ello, puede haber un servicio adicional que solo supervisa (no desencadena) los servicios que deben ejecutarse. De esta forma podemos monitorear para una ejecución exitosa, pero también podemos agregar nuevos servicios a pedido.

Desde el punto de vista de la arquitectura de microservicio, ambos son posibles y desde la práctica, se debe favorecer la coreografía debido a que esta implica un mayor grado de libertad en la forma en que se pueden manejar las cosas. Cuando se produce un determinado evento, cada servicio que escucha el evento se activa para hacer su parte. En caso de agregar un nuevo servicio, esté solo necesita escuchar este evento y también hacer su parte si el evento ocurre. En la actualidad, los servicios de IoT a menudo se combinan utilizando un estilo de orquestación, porque es más fácil de implementar y los protocolos como HTTP no tienen soporte nativo para la comunicación basada en eventos. Los servicios básicos se modelan como independientes, usando comunicación basada en eventos.

Los servicios de valor agregado deben vigilar que todos los servicios para una aplicación en particular estén presentes y escuchen los eventos correspondientes. Por lo tanto, la aplicación puede monitorear, si un evento inicial hace que se ejecuten todos los servicios relacionados. Si un servicio no responde a ese evento, el servicio de valor agregado puede tomar medidas correctivas.

2.2.2. Internet de las Cosas y Microservicios.

Con el continuo desarrollo y evolución de *Internet of Things* (IoT), la aplicación monolítica se vuelve mucho más grande en escala y aún más compleja en su estructura. Esto conduce a una pobre escalabilidad, extensibilidad y mantenibilidad, de ahí la arquitectura de microservicios se ha introducido en el campo de la aplicación IoT, debido a su flexibilidad, ligereza y acoplamiento flexible (Sun et al., 2017). La Figura 2.7 representa una arquitectura de Microservicios para IoT propuesta por Uviase and Kotonya (2018). La introducción de la capa de coordinación del servicio es para facilitar la identificación, clasificación, combinación y coreografía del servicio. Además, muestra varios microservicios que implementan las diversas interfaces requeridas (por ejemplo, MQTT, REST, etc.) que son proporcionadas por los elementos de detección física. Por ejemplo, si se solicita un informe desde la capa de la aplicación, realiza una llamada a la API *Gateway* que llama a la API REST apropiada. Un framework arquitectónico para IoT también será ventajoso desde el punto de vista de los

desarrolladores. A medida que más dispositivos se conectan a Internet, existe la necesidad de un marco de integración altamente escalable, extensible y tolerante a fallas. Adoptar la arquitectura de microservicio en IoT para remodelar los marcos de integración mejorará la confiabilidad de los sistemas de IoT.

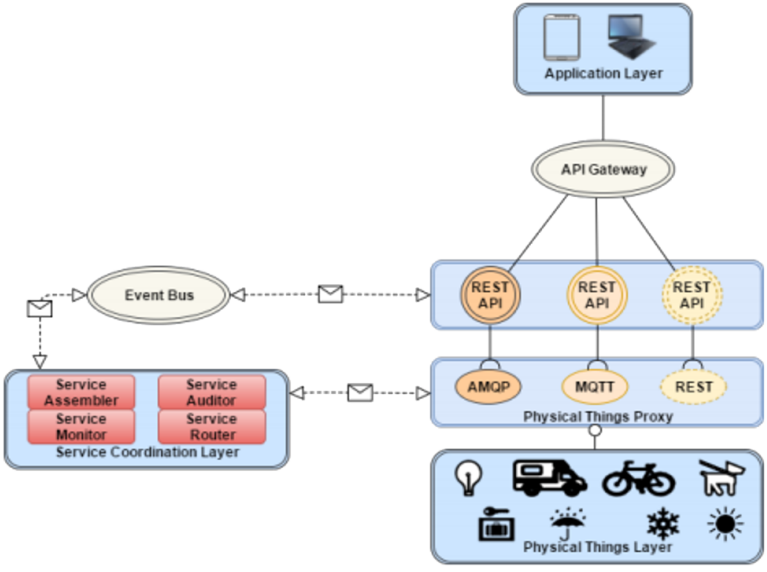


Figura 2.7: Arquitectura de Microservicio para IoT (Fuente: Butzin et al. (2016))

2.3. Metodologías Ágiles

Según Canós and Letelier (2012), una de las cualidades más destacables en una metodología ágil es su sencillez, tanto en su aprendizaje como en su aplicación, reduciéndose así los costos de implantación en un equipo de desarrollo. Esto ha llevado hacia un interés creciente en las metodologías ágiles. En la Tabla 2.1 se muestran las principales diferencias de las metodologías ágiles con respecto a las tradicionales ("no ágiles"). Estas diferencias que afectan no sólo al proceso en sí, sino también al contexto del equipo así como a su organización.

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo).	Impuestas externamente .
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Muchos artefactos.
Pocos roles.	Muchos roles.
Menos énfasis en la arquitectura del software.	La arquitectura del software es esencial y se expresa mediante modelos.

Tabla 2.1: Diferencias entre metodologías ágiles y no ágiles (Canós and Letelier, 2012).

Hoy en día, las organizaciones están utilizando metodologías ágiles para el desarrollo de aplicaciones; debido a que es un entorno de desarrollo rápido y también está en una escala mucho más grande después de la invención de la nube y las tecnologías distribuidas (Sharma, 2017). Según Rahimian and Ramsin (2008), las metodologías ágiles mejoran la flexibilidad y la productividad del desarrollo de software, proporcionando medios para adaptarse a los cambios en los requisitos y el medio ambiente, y también para aprender de las experiencias de desarrollo. Para soportar la entrega temprana y rápida de software en funcionamiento, estas metodologías usan motores de desarrollo iterativo-incremental para producir artefactos tangibles para el cliente.

El desarrollo de software ágil se basa en doce principios establecidos (Manifiesto, 2001):

- La mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
- Se acepta que los requisitos cambien, incluso en etapas tardías del desa-

rrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

- Se entrega software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
- Los responsables de negocio y los desarrolladores trabajan juntos de forma cotidiana durante todo el proyecto.
- Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
- El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
- El software funcionando es la medida principal de progreso.
- Los procesos ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios deben ser capaces de mantener un ritmo constante de forma indefinida.
- La atención continua a la excelencia técnica y al buen diseño mejora la agilidad.
- La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
- Las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados.
- A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

2.3.1. Metodologías ágiles y roles involucrados

Existen varias metodologías ágiles. Molina (2014) hace un estudio de estas metodologías que definen diferentes roles para sus actividades y procesos. A continuación se presenta un resumen de su propuesta, el mismo que resulta de interés en este trabajo de titulación.

A. *Agile Project Management* (APM)

Propone una nueva visión de gestión de proyectos ágiles y adaptativos. Así mismo, plantea que los procesos deben ajustarse a los objetivos de negocio, si ésos son repetibles y predecibles, entonces un proceso prescriptivo es lo más adecuado, pero si los objetivos de negocio son innovadores, entonces el marco de trabajo de los procesos debe ser ágil, flexible y adaptable.

Los roles involucrados se muestran a continuación:

- Patrocinador ejecutivo. Es la persona o grupo que lidera el producto.
- Gestor de proyectos. Es la persona que lidera el equipo que libera o entrega el producto.
- Gestor de producto. Es la persona que lidera el equipo que determina los resultados a entregar.
- Ingeniero jefe. La persona que guía los aspectos técnicos del producto.
- Gestores. Un grupo que puede estar a cargo de organizaciones participantes.
- Equipo de cliente. Grupo que determinan las características a ser desarrolladas y priorizadas.
- Equipo de proyecto. Grupo que libera o entrega los resultados.
- Proveedores. Organizaciones externas o personas físicas que proveen servicios o componentes.
- Gobierno. Entes reguladores que requieren información, reportes, certificaciones y otros documentos legales.

B. Crystal

La familia de métodos Crystal define un código de color para establecer la complejidad de la metodología: si es más oscuro entonces el método es más pesado. Cuanto más crítico es el sistema, más rigor se necesita. Incluye un conjunto de principios para adaptar las diferentes metodologías según las circunstancias del proyecto.

Los métodos Crystal no prescriben las prácticas de desarrollo, herramientas o productos, y como mencionamos se pueden combinar con otras metodologías.

C. *Dynamic System Development Methods*

El método de desarrollo de sistemas dinámicos (DSDM) proporciona un marco de trabajo completo de controles para el desarrollo rápido de aplicaciones y lineamientos para su utilización y se puede complementar con otras metodologías.

DSDM define hasta quince roles distintos, pero los más importantes son:

- Programadores y Programadores Senior: Senior indica el nivel de liderazgo en el equipo. Se cubren todos los roles de desarrollo.
- Coordinador técnico: Define la arquitectura del sistema y es responsable de la calidad técnica del proyecto, control técnico y configuración del sistema.
- Usuario embajado: Proporciona el conocimiento de la comunidad de usuarios y difunde información sobre progresos. Adicionalmente, puede existir un Usuario Asesor (Advisor) con otros puntos importantes de los usuarios, puede ser personal TI o un auditor funcional.
- Visionario: Es un usuario participante que tiene la percepción más exacta de objetivos del sistema y proyecto. Asegura que los requerimientos esenciales se cumplan y que el proyecto vaya en una dirección adecuada a éstos.
- Patrocinador ejecutivo: Es la persona de la organización que detenta autoridad y responsabilidad financiera.
- Facilitador: Es responsable de administrar el progreso del taller y motor de la preparación y comunicación.
- Escriba: Registra los requerimientos, acuerdos y decisiones alcanzadas en las reuniones, talleres y sesiones de prototipado.

D. *Extreme Programing* (XP)

XP se basa en cuatro valores: comunicación, simplicidad, retroalimentación y coraje. El proceso está constituido por seis fases:

- Fase de exploración. En esta fase los clientes escriben las tarjetas de historia que serán incluidas en la primera versión. Cada una de estas tarjetas describirán una funcionalidad que será agregada al programa.
- Fase de planificación. Se define la prioridad de las distintas historias y se acuerda el contenido de la primera entrega del proyecto.

- Fase de iteraciones. La planificación divide el tiempo en varias iteraciones. Los usuarios deciden qué historias se realizarán en cada iteración, ya que la primera entrega suele contener toda la arquitectura del sistema. Las pruebas funcionales son creadas por el cliente y se ejecutan al término de cada iteración.

- Fase de producción. Se ejecutan una serie de pruebas extra, de rendimiento, de funcionamiento necesarias antes de entregar el producto al cliente. Si se deben hacer cambios debe decidirse si incluirlos en esta entrega o en las próximas.

- Fase de mantenimiento. Liberada la versión al cliente, el proyecto se debe mantener en el entorno siempre que sigan habiendo iteraciones en esa fase.

- Fase de cierre de proyecto. Los clientes ya no tienen historias para ser implementadas, por lo que es necesario estar seguros que estamos cumpliendo con todas las necesidades de los clientes, y aspectos como fiabilidad, rendimiento. La documentación del proyecto se hace aquí, ya que no habrá más cambios.

Los roles XP son pocos (Duarte and Rojas, 2008):

- Un cliente: Escribe historias y pruebas de aceptación;

- Programadores en pares: Escribe las pruebas unitarias y produce el código del sistema.

- Tester (Encargados de las pruebas): Ayudan al cliente a desarrollar pruebas funcionales, difunde los resultados en el equipo y es responsable de las herramientas de soporte de las pruebas.

- Consultores técnicos: Un miembro externo con conocimiento de algún tema necesario para el proyecto.

- Tracker (Encargado del seguimiento): Proporciona retroalimentación al equipo, realiza el seguimiento del proceso en cada iteración.

- Gestor o Gran jefe. Es el vínculo entre clientes y programadores, ayuda a que el equipo trabaje efectivamente proporciona las condiciones adecuadas. Su labor esencial es de coordinación.

E. SCRUM

Scrum es adaptativo, ágil, auto-organizante y con pocos tiempos muertos. Scrum no está concebido para ser utilizado independientemente, sino en combinación con otras metodologías. Se enfoca en valores y prácticas de gestión, sin mencionar requerimientos, implementación u otros temas técnicos. Utiliza técnicas de control de procesos que aplica en gestión y control de proyectos.

Los roles en Scrum son seis:

- Scrum Master: Debe interactuar con el equipo, el cliente y los gestores. Garantiza el funcionamiento de los procesos y la metodología. Debe ser miembro del equipo y trabajar a la par. Coordina los encuentros diarios y se encarga de eliminar obstáculos.
- Propietario del producto: Es el responsable oficial del proyecto, gestión, control y visibilidad del product backlog. Elegido por el Scrum Master, el cliente y los ejecutivos.
- Equipo de desarrollo: Tiene autoridad para reorganizarse y definir acciones necesarias o sugerir eliminar problemas para cumplir con los objetivos del sprint.
- El cliente: Participa en la creación del Product Backlog.
- El gestor: Toma las decisiones finales, participa en la elección de objetivos y requisitos. Selecciona el Propietario del producto.
- El usuario

2.3.2. Desarrollo ágil aplicado al enfoque DevOps

Según manifiesta Httermann (2012), los proyectos de software a menudo pasan por las siguientes fases:

I. **Fase inicial:** En este intervalo, se desarrolla la visión del sistema, se define el alcance del proyecto y se justifica el caso de negocio.

II. **Elaboración:** En este intervalo, se reúnen y definen los requisitos, se identifican los factores de riesgo y se inicializa una arquitectura del sistema.

III. **Construcción:** En este intervalo, el software se construye, programa y prueba.

IV. **Transición:** En este intervalo, el software se entrega al usuario.

v. **Operaciones:** En este intervalo, el software está disponible para el usuario y es mantenido por las operaciones.

Como se observa en la Figura 2.8, el desarrollo de software ágil abarca el proceso desde el inicio hasta la transición. DevOps abarca el proceso desde la elaboración hasta las operaciones, y a menudo incluye departamentos tales como recursos humanos y finanzas. Los esfuerzos ágiles a menudo terminan en la fase de transición del desarrollo a las operaciones. La entrega de software (es decir, enviar el software a la producción y ponerlo a disposición de los usuarios finales) está cubierto por DevOps, el mismo que es abordado a detalle en la siguiente subsección y que proporciona patrones para fomentar la colaboración entre los interesados del proyecto y utiliza procesos, así como herramientas para agilizar el proceso de entrega del software y reducir el tiempo total del ciclo (Httermann, 2012).

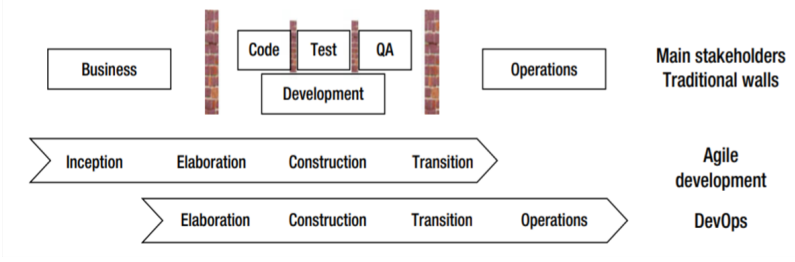


Figura 2.8: Arquitectura de Microservicio para IoT (Fuente: (Httermann, 2012)).

2.4. DevOps

A pesar de que no existe una definición clara de lo que es DevOps, basados en el estudio de Dyck et al. (2015), quién evalúa y discute las definiciones existentes, descartando que sea una metodología y definiéndolo como sigue:

"DevOps es un enfoque organizacional, que enfatiza la empatía y la colaboración multifuncional dentro y entre los equipos, especialmente para el desarrollo y las operaciones de TI, en las organizaciones de desarrollo de software de alta calidad, para operar sistemas resilientes y acelerar la entrega de cambios" Dyck et al. (2015).

Definición que concuerda con lo establecido por Ebert et al. (2016), quien trata de dar una explicación clara de DevOps, su ciclo de vida y su automatización mediante herramientas. DevOps integra los dos mundos de desarrollo y operaciones, utilizando desarrollo automatizado, implementación y monitoreo

en su infraestructura. Es un cambio organizativo en el que, en lugar de grupos divididos distribuidos que realizan funciones por separado, los equipos multifuncionales (*cross-functional*) trabajan en entregas continuas de características operativas. La Figura 2.9 muestra el proceso genérico, que tiene como objetivo integrar mejor los procesos comerciales de desarrollo, producción y operaciones con la tecnología adecuada (Ebert et al., 2016).

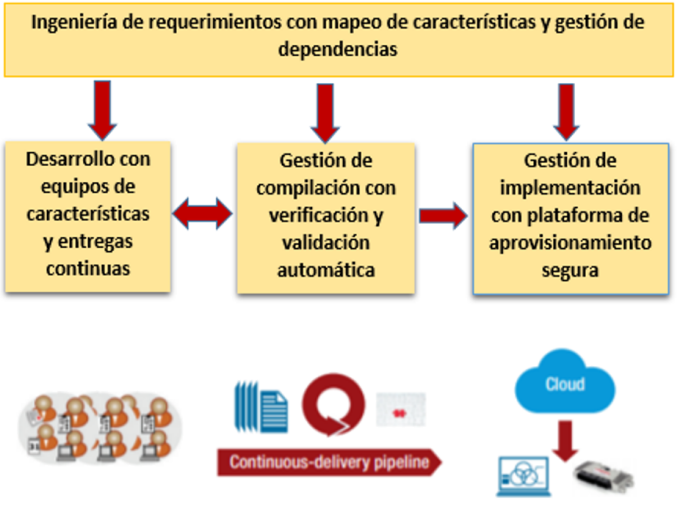


Figura 2.9: Proceso genérico de producción y entrega de DevOps. Su objetivo es integrar mejor los procesos comerciales de desarrollo, producción y operaciones con la tecnología adecuada (Fuente: (Ebert et al., 2016)).

2.4.1. Equipos de trabajo en DevOps

Como explica (Kim et al., 2016), en DevOps se requiere de diferentes equipos entre los cuales tenemos:

- **Dueño del producto:** Define el conjunto de funcionalidades que tendrá el nuevo servicio.
- **Equipo de desarrollo:** Equipo de personas encargadas de desarrollar las funcionalidades de la aplicación.

- **Aseguramiento de calidad:** Equipo responsable de asegurarse que las funciones a proporcionar son las deseadas.
- **Operaciones:** Equipo responsable de mantener el entorno de producción.
- **Seguridad de la información:** Equipo encargado del aseguramiento de los sistemas y los datos.
- **Gestores de liberaciones:** Responsables de gestionar y coordinar los procesos de liberación a producción.
- **Ejecutivo tecnológico:** Responsable de asegurarse que el objeto de valor cumpla o supere los requerimientos del cliente.

DevOps recomienda la división vertical de los miembros del proyecto en pequeños equipos multifuncionales que también se adaptan bien a los microservicios. Cada equipo es responsable de un servicio y consta de personas con diferentes habilidades, como desarrollo y operaciones, y cooperan desde el comienzo del proyecto para crear más valor para los usuarios finales de ese servicio en particular a través de la puesta en producción más frecuente de nuevas funciones, por lo tanto; eliminar los gastos generales de transición que existían en la formación del equipo horizontal. Además, como cada equipo se enfoca en un servicio en particular, la capacidad de mantenimiento y la comprensión del código de cada servicio es mucho más alta, y se pueden agregar nuevos miembros a los equipos con una curva de aprendizaje más baja (Dyck et al., 2015).

Además, debería formarse un equipo central compuesto por representantes del equipo de cada servicio que es responsable de las capacidades compartidas. Este equipo tiene una visión general de las interacciones del servicio en el sistema y está a cargo de cualquier decisión arquitectónica crítica. Además, cualquier refactorización entre servicios que implique la transferencia de funcionalidades entre servicios y la actualización de las reglas correspondientes en el servidor perimetral se maneja a través de este equipo (Balalaie et al., 2016).

2.4.2. Ciclo de vida de DevOps

Como menciona Sharma and Coyne (2013), DevOps se basa en 4 fases: Planificación o Dirección, Desarrollo y Pruebas, Despliegue y Operaciones. En la Figura 2.10 se muestran dichas fases junto con las diferentes actividades que abarcan. A diferencia de las otras prácticas de desarrollo, con DevOps se consigue la liberación de nuevas funcionalidades en breves plazos de tiempo además de la obtención de una rápida retroalimentación.

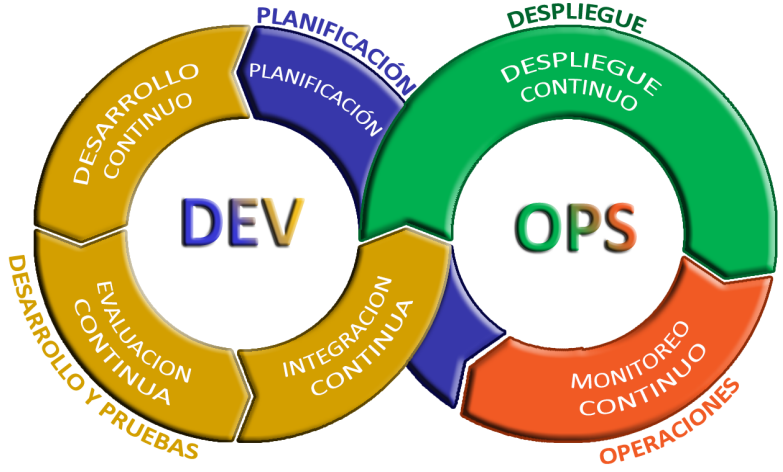


Figura 2.10: Arquitectura de referencia de DevOps (Fuente: Elaboración propia).

A. Planificación

Durante esta fase se establecen las nuevas funcionalidades o cambios a desarrollar durante este ciclo, para ello; DevOps ayuda a los equipos a establecer objetivos de negocios en colaboración y a cambiarlos continuamente en función de los comentarios de los clientes, mejorando así la agilidad y los resultados comerciales.

B. Desarrollo continuo

En esta fase se construye el software que contiene las nuevas funcionalidades requeridas, una entrega de software involucra diferentes equipos interfuncionales. Gracias al desarrollo colaborativo estos equipos pueden trabajar juntos al proporcionar un conjunto común de prácticas y una plataforma común que pueden usar para crear y entregar software. Por ello, el equipo de desarrollo emplea herramientas que faciliten la gestión y el versionamiento de código.

Luego de contar con el código desarrollado se requiere generar archivos ejecutables es por ello que se requiere herramientas que faciliten la gestión de dependencias y la compilación del código.

C. Evaluación continua

La evaluación continua recomienda realizar pruebas de manera anticipada y continua durante todo el ciclo de vida, esto resulta en costos reducidos, ciclos de prueba más cortos y una retroalimentación continua sobre la calidad. Este proceso hace hincapié en la integración de las actividades de desarrollo y prueba para garantizar que la calidad esté integrada lo más temprano posible en el ciclo de vida y no quede algo para más adelante. Esto se facilita mediante la adopción de capacidades como pruebas automatizadas y virtualización de servicios.

D. Integración continua

Durante esta fase un equipo de desarrollo integra su trabajo con el de otros miembros del equipo de desarrollo, para que, luego sea sometido a pruebas de integración. El objetivo de esta actividad es exponer el producto a riesgos de integración. Esta fase corresponde a la primera fase de operación es por ello que desde esta fase no se produce ni gestiona la generación del código, en esta fase; a través de herramientas de automatización se realiza las siguientes actividades:

- Comprobación del código en el repositorio de código.
- Ejecución de pruebas unitarias.
- Las nuevas funcionalidades se integran con las preexistentes para buscar algún problema de integración.

Durante esta fase se puede determinar algún desperfecto en el código en cualquiera de las fases anteriores, si el código supera las pruebas de integración puede ser implementado en el área de producción.

E. Despliegue continuo

Esta fase lleva el proceso de integración continua al siguiente paso permitiendo la liberación y despliegue de software en un entorno de producción de manera eficiente y automatizada. El objetivo perseguido en esta fase es permitir lanzar nuevas funcionalidades a los clientes y usuarios lo más pronto posible reduciendo la intervención humana. Durante esta fase se deben considerar dos tipos de herramientas involucradas con el entorno de despliegue:

- Herramientas que permitan gestionar el entorno, realizar y superar pruebas relacionadas al entorno (como pueden ser pruebas de rendimiento, funcionalidad o seguridad).

- Herramientas para la gestionar la configuración de aplicaciones en un entorno de despliegue.

F. Monitoreo continuo

El monitoreo continuo proporciona datos y métricas a las operaciones, al control de calidad, al personal de desarrollo de líneas de negocio y a otras partes interesadas sobre las aplicaciones en las diferentes etapas del ciclo de entrega.

2.4.3. Automatización de DevOps

De acuerdo con Ebert et al. (2016) las herramientas son obligatorias para automatizar DevOps. Las entregas de calidad con un ciclo de tiempo corto requieren un alto grado de automatización. Por lo tanto, elegir las herramientas adecuadas para el proyecto es importante para el éxito del sistema de software. La Tabla 2.2 enumera las herramientas de automatización que Ebert et al. (2016) analiza, en cada una de las fases que se describen a continuación.

Herramienta	**Fase de DevOps**	**Tipo Herramienta**	**Formato de Configuración**	**Lenguaje**	**Licencia**
Ant	*Build*	*Build*	XML	Java	Apache
Maven	*Build*	*Build*	XML	Java	Apache
Rake	*Build*	*Build*	Ruby	Ruby	MIT
Gradle	*Build*	*Build*	Basado en Groovy	Java y lenguaje de dominio específico basado en Groovy	Apache
Jenkins	*Build*	*Continuous integration*	UI	Java	MIT
TeamCity	*Build*	*Continuous integration*	UI	Java	Comercial
Bamboo	*Build*	*Continuous integration*	UI	Java	Comercial
Puppet	*Deployment*	*Configuration management*	DSL similar a JSON (JavaScript Object Notation)	Ruby	Apache
Chef	*Deployment*	*Configuration management*	DSL basado en Ruby	Ruby	Apache
Ansible	*Deployment*	*Configuration management*	YAML	Python	GPL (Licencia pública general GNU)
Loggly	*Operations*	*Operations*	-	Basado en Cloud	Comercial
Graylog	*Operations*	*Logging*	-	Java	Opensource
Nagios	*Operations*	*Monitoring*	-	C	Opensource y GPL
New Relic	*Operations*	*Monitoring*	-	-	Comercial
Cacti	*Operations*	*Monitoring*	-	PHP	GPL

Tabla 2.2: Herramientas de Automatización de DevOps

A. Compilación (Build)

En esta fase de DevOps, las herramientas deben soportar flujos de trabajo rápidos. Dentro de este flujo de trabajo se consideran dos tipos de herramientas:

- Las herramientas de compilación ayudan a lograr una iteración rápida, reduciendo las tareas manuales que consumen mucho tiempo.
- Las herramientas de integración continua combinan el código de todos los desarrolladores y comprueban si hay código roto, lo que mejora la calidad del software.

La automatización de la compilación es crucial para el entorno DevOps, tanto en el pequeño como en el grande. Con un desarrollo ágil y fluido, las herramientas de desarrollo también se han convertido en herramientas para administrar el desarrollo de software y el ciclo de vida del servicio, lo que implica compilar código, administrar dependencias, generar documentación, ejecutar pruebas o implementar una aplicación en diferentes entornos.

B. Integración Continua (CI)

Las herramientas de CI integran el trabajo de los desarrolladores tan pronto como sea posible. De esta manera, el sistema se prueba constantemente. Aceptar CI no siempre es fácil y directo. Por ejemplo, en el desarrollo de sistemas integrados, las pruebas son un reto porque la construcción y las pruebas en el hardware de destino no siempre son posibles y, por lo tanto, deben simularse. Además, las limitaciones del hardware afectan la velocidad de prueba y, por lo tanto, el tiempo de ciclo.

C. Despliegue

Durante esta fase, el cambio más importante es tratar la infraestructura como código. Con este enfoque, la infraestructura puede compartirse, probarse y controlarse por cada versión. Las herramientas de gestión de la configuración pueden reducir la provisión de la producción y la complejidad del mantenimiento de la configuración al tiempo que permiten la recreación del sistema de producción en las máquinas de desarrollo. Dichas herramientas son un importante habilitador de DevOps. Para lograr una entrega continua de sistemas integrados, los dispositivos deben estar conectados, por lo que la comunicación de máquina a máquina y una arquitectura de IoT son necesarias.

D. Operaciones

DevOps también afecta las operaciones de infraestructura. Por lo tanto, necesita herramientas para mantener la estabilidad y el rendimiento de su infraestructura.

E. Herramientas de registro

El registro a veces se considera un reemplazo para la depuración, pero desde un punto de vista DevOps, los rastros de la depuración son necesarios para administrar aplicaciones. La regla es registrar todo y determinar el nivel del registro (por ejemplo: fatal error, advertir, información, rastreo o depuración). Para las aplicaciones que no son muy complejas, puede usar las herramientas estándar (Java, *java.util.logging*). Para proyectos más complejos, puede usar marcos como Log4j o sistemas de administración de registros.

F. Herramientas de monitoreo

Permiten que las organizaciones identifiquen y resuelvan los problemas de la infraestructura de TI antes de que afecten a los procesos comerciales críticos. Estas herramientas supervisan aspectos del sistema tales como la carga de la CPU, la asignación de RAM, las estadísticas de tráfico de red, el consumo de memoria y la disponibilidad de espacio libre en el disco. Rastrean continuamente a los administradores de monitoreo y alertas del sistema cuando detectan problemas para que los administradores puedan tomar medidas correctivas.

2.4.4. DevOps aplicando microservicios con metodología "12 factores de aplicaciones".

Como describe Wiggins (2011) *The Twelve-Factor App* es una metodología basada en 12 factores y que permite construir aplicaciones SaaS escritas en cualquier lenguaje de programación, y cualquier combinación de *"backing services"* (bases de datos, memoria caché, etc). Las aplicaciones desarrolladas bajo esta metodología cuentan con las siguientes características:

- Usan formatos declarativos.
- Son apropiadas para desplegarse en plataformas en la nube.
- Minimizan las diferencias entre entornos de desarrollo y producción, permitiendo el despliegue continuo.
- La aplicación puede escalar sin cambios significativos para las herramientas, la arquitectura y prácticas de desarrollo.

Cada factor propuesto se examina a continuación junto con su vinculación con las fases de DevOps.

A. Un código base sobre el que hacer el control de versiones y múltiples despliegues

The Twelve-Factor App recomienda el uso de un código base por aplicación, en arquitectura de microservicios se recomienda un código base por microservicio. Un código base es cualquier repositorio (un sistema de control de versiones) o cualquier conjunto de repositorios que comparten un *commit* raíz (en un sistema de control de versiones descentralizado). Esto puede ser aplicado en DevOps mediante herramientas de permitan que permitan el versionamiento de código tales como Git.

B. "Declarar explícitamente las dependencias"

Una aplicación no debe depender de la existencia de paquetes instalados en el sistema por lo que se recomienda utilizar un gestor de dependencias. Las dependencias de código pueden ser cubiertas con herramientas de compilación estándar como Maven o Gradle durante la fase de compilación en DevOps. En caso de requerir servicios como por ejemplo bases de datos o servicios externos, en entornos no contenerizados se recomienda utilizar herramientas de gestión de configuración tales como Chef o Puppet para instalar dependencias de sistema. En entornos contenerizados esta gestión se realiza mediante *Dockerfiles*.

C. Guardar la configuración en el Entorno

Una configuración de aplicación es todo lo que puede variar entre despliegues, lo cual incluye:

- Recursos que manejan base de datos.
- Credenciales para servicios externos.
- Valores de despliegue como por ejemplo el nombre canónico del equipo.

Para ello se recomienda:

- Utilizar variables de entorno que puedan ser cambiables en runtime.
- Utilizar archivos sin versión controlada (*non-version-controlled files*) .env para despliegues locales. La herramienta Docker soporta la carga de esos archivos en runtime.

D. Tratar los *"Backing services"* como recursos conectables

Un *backing service* es cualquier recurso que la aplicación puede consumir a través de la red como parte de su funcionamiento habitual. En microservicios todo servicio externo se trata como un recurso, incluyendo otros servicios. Esto garantiza que cada servicio es completamente portable y débilmente acoplado con otros recursos del sistema.

E. Separar completamente la etapa de construcción de la etapa de ejecución

Aplicando este factor el código base se separa la aplicación de las fases de construcción, integración, distribución y ejecución. La fase de construcción un repositorio de código se convierte en un paquete ejecutable llamado construcción. En DevOps esto se realiza en la fase de construcción en la actividad "Desarrollo continuo". La fase de integración en DevOps corresponde a la automatización de pruebas realizadas en "Integración Continua" la cual puede ser automatizada mediante herramientas de Integración continua.

Durante la fase de distribución se usa la construcción creada en la fase de construcción y se combina con la configuración del despliegue actual. Por tanto, la distribución resultante contiene tanto la construcción como la configuración y está lista para ejecutarse inmediatamente en el entorno de ejecución. En DevOps se puede automatizar este proceso aplicando herramientas empleadas en "Despliegue continuo" (correspondiente a tareas de despliegue y configuración).

F. Ejecutar la aplicación como uno o más procesos sin estado

En microservicios, la aplicación no debe tener un estado. Esto permite el escalamiento horizontal de la aplicación añadiendo más instancias del servicio que no almacena ningún dato de estado.

G. Cada servicio maneja sus propios datos

Esta es una modificación del factor "Asociación de puertos", recomienda permitir el acceso a los datos persistentes únicamente a través de la API de servicio. Esto evita contratos de servicio implícitos entre microservicios.

H. Escalar mediante el modelo de procesos

En una arquitectura de microservicios, es posible el escalado horizontal de cada servicio de forma independiente. Con servicios contenerizados esto puede obtenerse de forma gratuita.

I. Hacer el sistema más robusto intentando conseguir inicios rápidos y finalizaciones seguras

Las instancias deben ser desechables para que puedan iniciarse, detenerse y redistribuirse rápidamente y sin pérdida de datos. Los servicios implementados en contenedores Docker satisfacen este requisito de forma automática, ya que es una característica inherente de los contenedores que pueden detenerse e iniciarse de manera instantánea. Almacenar datos de estado o sesión en colas u otros servicios de respaldo asegura que una solicitud se maneje a la perfección en el caso de un bloqueo del contenedor.

J. Mantener desarrollo, preproducción y producción tan parecidos como sea posible

Se recomienda tener los entornos de desarrollo y producción lo más idénticos posibles para reducir el riesgo de bugs que aparecen en algunos entornos. Esto puede lograrse gracias a los sistemas de gestión de paquetes para instalación de *backing services*. De igual manera, las herramientas de gestión de la configuración (empleadas en despliegue continuo en DevOps) como Chef y Puppet combinadas con entornos virtuales ligeros como Docker o Vagrant permiten a los desarrolladores ejecutar entornos locales que son muy parecidos a los entornos de producción. El coste de instalar y usar estos sistemas es bajo comparado con el beneficio que se puede obtener de la paridad entre desarrollo y producción.

K. Tratar los historiales como una transmisión de eventos

En lugar de incluir código en un microservicio para enrutar o almacenar registros, use una de las soluciones de administración de registros, con DevOps se puede implementar mediante las herramientas de loggin.

L. Ejecutar las tareas de gestión/administración como procesos que solo se ejecutan una vez

Es frecuente que los desarrolladores quieran ejecutar procesos de administración o mantenimiento una sola vez (por ejemplo migraciones de bases de datos, ejecución de código arbitrario o scripts incluidos en el repositorio de la aplicación). Se recomienda en un entorno de producción, ejecutar las tareas administrativas y de mantenimiento por separado de la aplicación. Los contenedores hacen que esto sea muy fácil, ya que se puede desplegar un contenedor solo para ejecutar una tarea y luego cerrarlo.

2.5. Ingeniería en Requerimientos

Según Wohlin et al. (2005) la Ingeniería de Requerimientos se acepta como una de las etapas más cruciales en el diseño y desarrollo de software, ya que aborda el problema crítico de diseñar el software adecuado para el cliente. La ingeniería de requerimientos se refiere a todas las actividades del ciclo de vida relacionadas con los requisitos, esto incluye principalmente reunir, documentar y gestionar los requisitos. Entre las principales prácticas de la gestión de requerimientos incluyen Sillitti and Succi (2005).

2.5.1. Elicitación, especificación y modelado de requisitos

Implica comprender las necesidades de los interesados, obtener requisitos, modelar y recopilarlos en un repositorio. Esta es una etapa importante en el desarrollo de software. Sin embargo, los requisitos tienden a ser incompletos e inconsistentes. Por lo tanto, siempre hay margen de mejora en estas actividades.

2.5.2. Priorización

Esta práctica permite a los gerentes de proyecto planificar entregas del software dando prioridad a los requisitos que el cliente considera más importantes. Los requisitos se pueden priorizar teniendo en cuenta muchos aspectos diferentes. Wohlin et al. (2005) brinda una orientación acerca de estos aspectos y cuáles deberían ser tomados en cuenta. Un aspecto es una propiedad o atributo de un proyecto y sus requisitos que pueden usarse para priorizar los requisitos. Los aspectos comunes son importancia, penalización, costo, tiempo y riesgo. Al priorizar requisitos basados en un único aspecto, es fácil decidir cuál es el más deseable. Al involucrar otros aspectos, como el costo, los clientes pueden cambiar de opinión y los requisitos de alta prioridad pueden ser menos importantes si son muy costosos de satisfacer.

A menudo, los aspectos interactúan y los cambios en un aspecto pueden tener un impacto en otro aspecto. Por lo tanto, es esencial saber qué efectos pueden tener tales conflictos, y es vital no solo considerar la importancia al priorizar los requisitos sino también otros aspectos que afectan el desarrollo del software y la satisfacción con el producto resultante. Se pueden priorizar varios aspectos, y puede no ser práctico considerarlos a todos. Los que se deben considerar dependen de la situación específica, y algunos ejemplos de aspectos adecuados para los proyectos de software se describen a continuación. Los actores generalmente evalúan los aspectos en un proyecto (gerentes, usuarios, desarrolladores, etc.)

2.5.3. Dependencias requerimientos y análisis de impacto

A medida que los requerimientos cambian, éstos pueden afectar significativamente el proyecto de software; por ello, es necesario determinar qué cambios pueden ser aplicados sin influir drásticamente en las entregas del software.

2.5.4. Negociación de requisitos

En muchas situaciones, el conflicto es inherente a los requisitos; por lo tanto, deben negociarse entre las partes interesadas. La actividad de negociación de requisitos es una de las actividades más importantes en el desarrollo de software, ya que tiene un gran impacto en el producto final.

2.5.5. Garantía de calidad

El propósito de la garantía de calidad es establecer niveles de confianza razonables y realistas al redactar y gestionar los requisitos. Es importante que tanto los clientes como los desarrolladores estén involucrados en las actividades de aseguramiento de la calidad en Ingeniería de Requerimientos, ya que influyen en el éxito de un proyecto. Es importante destacar que la garantía de calidad de los requisitos no es solo una actividad en la fase de requisitos en los proyectos. La garantía de calidad debe abordarse a lo largo del ciclo de vida del software. Los requisitos deben rastrearse durante el desarrollo y la calidad debe asegurarse, por ejemplo, a través de inspecciones, revisiones y pruebas.

2.6. Diseño Dirigido por el Dominio (DDD)

La tendencia actual de crear aplicaciones, utilizando arquitecturas de microservicio se basa en el concepto de diseño impulsado por dominio (DDD) y entre los profesionales DDD es un enfoque ampliamente aceptado para crear aplicaciones (Steinegger et al., 2017). Como lo explica de la Torre (2017), el Diseño Dirigido por el Dominio (DDD) propone un modelado basado en la realidad de negocio con relación a sus casos de uso. En el contexto de la creación de aplicaciones, DDD hace referencia a los problemas como dominios. Describe áreas con problemas independientes como contextos delimitados, entendiendo que cada contexto delimitado está correlacionado con un microservicio y resalta un lenguaje común para hablar de dichos problemas. Este enfoque de desarrollo de software fue propuesto por Evans (2011) el cual considera al sistema como un conjunto de modelos que representan el modelo de negocio, para ello; se considera las siguientes definiciones:

- **Dominio:** Porción del diseño y la implementación que corresponde a la implementación de software, y que se conforma con un conjunto de subdominios. Involucra todos los aspectos de negocio donde intervendrá el sistema a desarrollar.
- **Subdominio:** Porción del sistema a desarrollar y que abstrae una porción del sistema, contiene un conjunto de servicios específicos.
- **Lenguaje Ubicuo:** Un lenguaje estructurado alrededor del modelo de dominio y utilizado por todos los miembros del equipo para conectar todas las actividades del equipo con el software.
- **Contexto:** Entorno en el cual una palabra o declaración adopta un determinado significado en base al dominio en el que se encuentra.
- **Límite de contexto:** Una descripción de un límite (típicamente un subsistema, o el trabajo de un equipo particular) dentro del cual se define y aplica un modelo en particular. Un Contexto Acotado es un límite conceptual donde un modelo de dominio es aplicable. Proporciona un contexto para el lenguaje ubicuo que habla el equipo y se expresa en su modelo de software cuidadosamente diseñado (Vernon, 2013).
- **Modelo:** Un sistema de abstracciones que describe aspectos seleccionados de un dominio y se puede usar para resolver problemas relacionados con ese dominio.

2.6.1. Aplicación de Diseño Dirigido por el Dominio en la Arquitectura de Microservicios

Microservicios es la unión de un conjunto de mejores prácticas de varias comunidades, la automatización operativa, la infraestructura programable, las comunidades de operaciones de desarrollo, las comunidades de la nube y las comunidades de integración, en el Diseño Dirigido por el Dominio, combina las mejores características en torno al diseño estratégico, contexto delimitado, subdominios, cómo separar sus dominios y cómo particionar un dominio problemático muy grande en dominios más pequeños para una mejor gestión (Thönes, 2015). Sin embargo, las opciones de DDD solo deben aplicarse en el caso de implementar microservicios complejos con reglas de negocio importantes. Las responsabilidades más sencillas, como el servicio CRUD, se pueden administrar con enfoques más sencillos (de la Torre, 2017).

La clave está en dónde situar los límites al diseñar y definir un microservicio. Los patrones de DDD le ayudan a comprender la complejidad del dominio. En el modelo de dominio de cada contexto delimitado, debe identificar y definir las

entidades, los objetos de valor y los agregados que modelan el dominio. Debe crear y perfeccionar un modelo de dominio que se encuentre dentro de un límite definido por su contexto. Y esto se hace claramente patente en la forma de un microservicio. Los componentes situados dentro de esos límites acaban siendo sus microservicios, aunque, en algunos casos, los contextos delimitados o los microservicios pueden estar compuestos de varios servicios físicos. El DDD afecta a los límites y, por lo tanto, a los microservicios.

2.6.2. Fases de Diseño Dirigido por el Dominio.

El diseño basado o dirigido por dominios tiene dos fases distintas, tácticas y estratégicas. En el diseño basado en dominios estratégico, se define la estructura a gran escala del sistema. Ayuda a garantizar que la arquitectura permanece centrada en las funcionalidades del negocio. El diseño basado en dominios tácticos proporciona un conjunto de modelos de diseño que puede usarse para crear el modelo de dominio. Estos modelos incluyen entidades, agregados y servicios de dominio. Los modelos ayudar a diseñar microservicios coherentes y con acoplamiento flexible (Wasson, 2017).

De cualquier forma que los modelos de software están diseñados, ya sea tácticamente, estratégicamente, el objetivo siempre será reflejar un lenguaje ubicuo limpio modelado en un contexto explícitamente limitado (Vernon, 2013).

A. Modelado Estratégico

Un contexto delimitado es un límite conceptual donde un modelo de dominio es aplicable. Proporciona un contexto para el lenguaje ubicuo que habla el equipo y se expresa en su modelo de software cuidadosamente diseñado, como se muestra en la Figura 2.11.

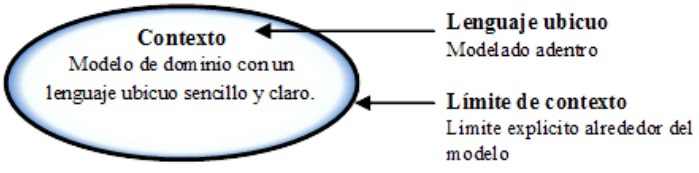


Figura 2.11: Diagrama que ilustra un contexto delimitado y la relevancia del lenguaje ubicuo (Fuente: (Vernon, 2013)).

B. Modelado Táctico

El Modelado Táctico se da dentro de un contexto delimitado utilizando los patrones de bloques de construcción de DDD, los siguientes conceptos han sido definidos por Evans (2011) quien introduce a DDD explicando sus conceptos y Vernon (2013) quien ofrece una vista general de la aplicación de DDD.

I. **Entidades**
Una entidad es un objeto con una identidad única que persiste en el tiempo, este identificador es único en el sistema y puede usarse para buscar la entidad o para recuperarla. Eso no significa, que el identificador siempre se exponga directamente a los usuarios. Podría ser un identificador único (GUID) o una clave principal de una base de datos. Algunas características relevantes de las entidades son:

- Una identidad puede abarcar varios contextos delimitados y puede durar más que la aplicación. Un ejemplo son los datos personales o los datos de cuenta de usuario.
- Los atributos de una entidad pueden cambiar con el tiempo. Por ejemplo, el nombre o la dirección de una persona pueden variar, pero sigue siendo la misma persona.
- Una entidad puede contener referencias a otras entidades.

II. **Objetos de valor**
Un objeto de valor no tiene identidad. Se define únicamente mediante los valores de sus atributos. Los objetos de valor también son inmutables. Para actualizar un objeto de valor, siempre hay que crear una nueva instancia que reemplace a la anterior. Los objetos de valor pueden tener métodos que encapsulan la lógica del dominio, pero esos métodos no deben afectar al estado del objeto. Ejemplos típicos de objetos de valor son los colores, las fechas y horas, y los valores de divisa.

III. **Agregados**
Un agregado define un límite de coherencia alrededor de una o varias entidades. Una entidad exacta en un agregado es la raíz. La búsqueda se realiza con el identificador de la entidad raíz. Cualquier otra entidad en el agregado es secundaria de la raíz y se hace referencia a ella siguiendo punteros desde esta. El propósito de un agregado es modelar las invariantes transaccionales.

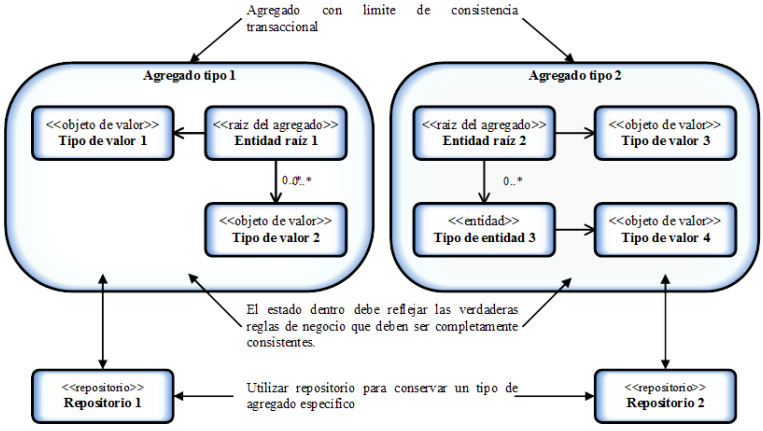


Figura 2.12: Dos tipos de agregado con sus propios límites de coherencia transaccional (Fuente: (Vernon, 2013)).

La Figura 2.12 muestra una explicación gráfica del patrón Agregado. Las cosas en el mundo real tienen redes complejas de relaciones. Por ejemplo, un sistema IoT tiene pacientes, cientes, los pacientes se monitorean con sensores, los sensores deben ser instalados por técnicos, y así sucesivamente.

La transaccionalidad depende de la aplicación, no del nivel de datos, el aplicar los valores invariables necesarios para el dominio. Un agregado se compone de una sola entidad o un clúster de entidades y objetos de valor que deben permanecer transaccionalmente consistentes a lo largo de toda la vida del agregado.

Comprender cómo modelar efectivamente los Agregados es muy importante, y además es una de las técnicas menos comprendidas entre los componentes básicos de DDD. Se conserva una instancia de un Agregado utilizando su Repositorio y luego se busca y se recupera de él. Los servicios sin estado deben usarse como se muestra en la Figura 2.13. Dentro del modelo de dominio para realizar operaciones comerciales que no encajan naturalmente como una operación en una Entidad o un Objeto de valor.

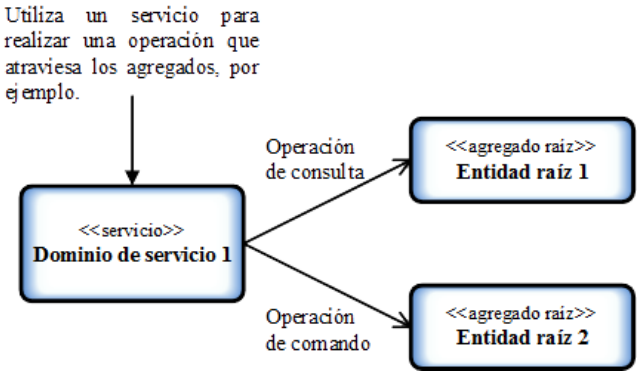


Figura 2.13: Los servicios de dominio llevan a cabo operaciones específicas de dominio, que pueden involucrar múltiples objetos de dominio (Fuente: (Vernon, 2013)).

IV. **Servicios de aplicación y de dominio**
En la terminología del diseño basado en dominios, un servicio es un objeto que implementa alguna lógica sin mantener ningún estado, aclarando además que los servicios de aplicación y de dominio no tienen relación en su significado con microservicios. Evans (2011) distingue entre servicios de dominio, que encapsulan la lógica del dominio, y servicios de aplicación, que proporcionan la funcionalidad técnica, como la autenticación del usuario o el envío de un mensaje SMS. Los servicios de dominio a menudo se utilizan para modelar el comportamiento que abarca varias entidades.

V. **Eventos de dominio**
Los eventos de dominio se pueden utilizar para notificar a otras partes del sistema cuando sucede algo. Los eventos de dominio son especialmente importantes en una arquitectura de microservicios. Dado que los microservicios se distribuyen y no comparten los almacenes de datos, los eventos de dominio proporcionan una manera de que los microservicios se coordinen entre sí.

A continuación, se revisan conceptos generales para el entendimiento de los componentes de DDD, Evans (2004) y Vernon (2013) definen claramente estos conceptos.

2.6.3. Diseño Dirigido por el Dominio basado en una Arquitectura de Capas.

En un sistema de software que aplica DDD se considera una división del sistema en capas, como se muestra en la Figura 2.14. Si bien como menciona Evans (2004) el diseño impulsado por el dominio requiere solo una capa en particular, la capa de dominio, que tiene toda la lógica de negocio, existen otras capas en un sistema de software y cada una posee responsabilidades en el sistema (Haywood, 2009).

- La capa de presentación tiene la lógica para representar una interfaz de usuario a través de la cual el usuario puede interactuar con la aplicación.
- La capa de aplicación tiene la lógica que gestiona el estado actual de la sesión y el nivel de conversación del usuario y posiblemente controla las interacciones entre esos estados.
- La capa de dominio contiene la lógica de negocios aplicable a todos los usuarios y todas las aplicaciones, es decir; a través de todo el dominio.
- La capa de persistencia (quizás etiquetada más generalmente como una capa de infraestructura) es donde se almacena el estado del objeto de dominio y quizás el estado lógico de la aplicación.

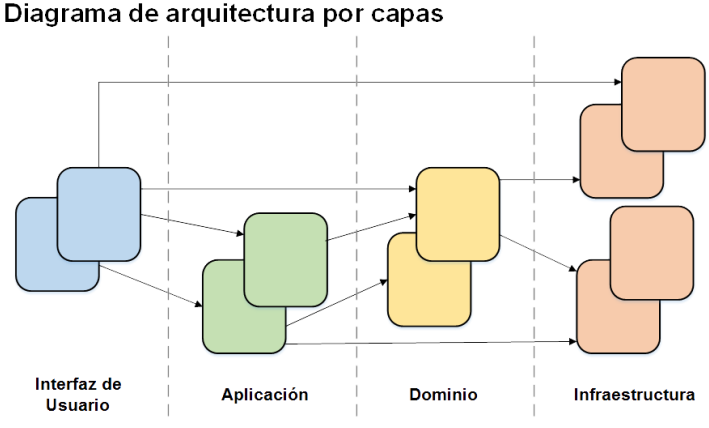


Figura 2.14: Arquitectura de Capas en DDD (Fuente: Elaboración propia).

Uno de los propósitos principales de una arquitectura es definir e idealmente restringir cómo interactúan estas capas (Haywood, 2009). Por último, tal como

lo sugiere Evans (2004). Aunque un desarrollador individual que entienda el diseño impulsado por dominio ganará valiosas técnicas de diseño y perspectiva, las mayores ganancias se obtienen cuando un equipo se une para aplicar un enfoque de Diseño Dirigido por el Dominio. Es por ello que es importante que DDD forme parte de la cultura organizacional de la empresa o del sistema de microservicios.

2.7. Diseño Dirigido por Capacidades de Negocio

El Diseño Dirigido por capacidades de negocio, como señala Richardson (2014), permite descomponer el sistema a desarrollar mediante la definición de servicios correspondientes a las capacidades empresariales. Una capacidad de negocio es un concepto del modelado de arquitectura de negocios. Es algo que hace una empresa para generar valor.

Las capacidades de negocio a menudo están organizadas en una jerarquía de niveles múltiples. Por ejemplo, una aplicación empresarial puede tener categorías de nivel superior, como el servicio de desarrollo del producto, el servicio de entrega del producto, la generación de demanda, etc.

Edwin Cabrera Alvarado
Paola Cárdenas Cárdenas

Capítulo 3

Estado del Arte

Este capítulo presenta una revisión sistemática de metodologías, soluciones existentes y procesos de software para la creación de sistemas basados en microservicios, para tener una idea de las propuestas metodológicas y procesos existentes para cada una de las áreas temáticas que aborda el presente trabajo de titulación. Se considera la revisión sistemática como un paso más del proceso investigativo, que tiene como objetivo la búsqueda, identificación y análisis de estudios relacionados al dominio, dicho proceso tiene como finalidad conocer cuáles son las aplicaciones y sistemas orientados a Ambientes de Vida Asistidos (AAL - *Ambient Assisted Living*) que han hecho uso de la arquitectura orientadas a servicios en el ámbito del Internet de las Cosas.

3.1. Introducción a las revisiones sistemáticas de la literatura

La revisión de la literatura es el proceso mediante la cual se consulta se extrae y recopila la información relevante sobre una interrogante de investigación específica, área temática o fenómeno de interés. (Cortés and León, 2005). Las Revisiones Sistemáticas son un diseño de investigación observacional y retrospectivo, que sintetiza los resultados de múltiples investigaciones primarias. Son parte esencial de la medicina basada en la evidencia por su rigurosa metodología, identificando los estudios relevantes para responder preguntas específicas de la práctica clínica. El término meta-análisis se reserva para la combinación numérica de los datos (Beltrán and Óscar, 2005).

Kitchenham et al. (2009) propone lineamientos que tienen como objetivo reflejar los problemas o necesidades específicas en la investigación de ingeniería de software, proponiendo tres fases comunes en revisiones sistemáticas: (i) planificación, (ii) conducción o ejecución de la revisión y (iii) difusión de resultados.

Los objetivos asociados a planeación de la revisión son:

- Identificar la necesidad de una revisión.
- Identificar claramente el propósito de la revisión.
- Especificar interrogantes o preguntas de investigación.
- Desarrollar un protocolo de revisión.
- Evaluar el protocolo de revisión.

Los objetivos asociados con la ejecución de la revisión son:

- Identificar la investigación.
- Seleccionar estudios primarios.
- Evaluar la calidad de los estudios primarios.
- Extraer los datos y monitorearlos.
- Sintetizar los datos extraídos.

Los objetivos asociados con la difusión de resultados de la revisión son:

- Especificar mecanismos de diseminación.
- Conformar el reporte principal.
- Evaluar el aporte.

3.2. Revisión sistemática de la literatura

El protocolo de revisión presentado a lo largo de este trabajo obedece a las guías planteadas por Kitchenham et al. (2009), en donde se establecen tres fases para la realización de una revisión sistemática de la literatura dentro de la Ingeniería de Software: (i) planificación, (ii) conducción o ejecución de la revisión y (iii) difusión de resultados.

3.2.1. Planificación de la revisión

En esta sección se presenta el protocolo de revisión a seguir. Según Kitchenham et al. (2009) "Antes de llevar a cabo una revisión sistemática, es necesario confirmar la necesidad de dicha revisión, tomando en cuenta que las actividades más importantes previas a la revisión son definir las preguntas de investigación que abordará la revisión sistemática y elaborar un protocolo de revisión, es decir; un plan que defina los procedimientos de revisión básicos". Esta fase presenta tres etapas (i) Identificación de la necesidad de la revisión, (ii) Formulación de las preguntas y subpreguntas investigación y (iii) Protocolo de búsqueda, descritas a continuación.

A. Identificación de la necesidad

Para identificar la necesidad de llevar a cabo una revisión sistemática que aporte significativamente a la contribución de este trabajo de titulación se ha llevado a cabo una búsqueda de revisiones sistemáticas de: microservicios, Internet de las Cosas, y *Ambient Assisted Living*. Otras revisiones sistemáticas encontradas han sido relevantes para identificar cuáles son las brechas investigativas en las que se enfocará la revisión sistemática planteada y definir la necesidad de realizar la misma.

Algunas revisiones sistemáticas abarcan el término de microservicios en ámbitos generales si hacer énfasis al área de dominio, en la cual los microservicios se están aplicando. En el caso de Garriga (2017), se realiza una revisión sistemática de microservicios para identificar el ciclo de vida de una aplicación basada en la arquitectura de microservicios considerando el diseño, implementación, ejecución, aspectos de calidad y aspectos organizacionales. El mapeo sistemático presentado por Vural et al. (2017) tiene como objetivo conocer las tendencias en torno a los microservicios, la motivación detrás de la investigación de microservicios, los estándares emergentes y las posibles lagunas de investigación, concluyendo así que el término microservicios apareció por primera vez en 2014, haciendo referencia en los estudios analizados en dicho mapeo sistemático, además Vural et al. (2017) concluye que "No hay suficientes estudios empíricos para aclarar muchos temas en discusión relacionados con los microservicios.

Además, no hay investigación específicamente dirigida a los puntos débiles de los microservicios, como las "transacciones distribuidas". No obstante en el mapeo sistemático realizado por Pahl and Jamshidi (2016), quien especifica que si bien no realizó un análisis sumativo concluyente, debido el reciente surgimiento del término microservicios a partir del año 2014, se puede determinar que los microservicios emergen como un estilo arquitectónico, pero que se extiende desde la "arquitectura de la etapa de diseño" hacia el despliegue y

las operaciones como un estilo de desarrollo continuo: la dimensión del "método". También parece que una parte significativa de los estudios revisados está intrínsecamente relacionada con los contenedores basados en la nube para la implementación y la gestión dinámica: la dimensión de "arquitectura dinámica".

El estudio de Kampmeijer et al. (2016) proporciona una revisión sistemática de la evidencia sobre el alcance del uso de las herramientas de ehealth y mhealth en cuidados de salud y prevención primaria entre los adultos mayores, realiza también una evaluación de las características de las herramientas ofrecidas para adultos mayores.

De lo que se conoce a través de las búsquedas preliminares, no existe una investigación enfocada a metodologías, técnicas y procesos sobre la integración e implementación de microservicios dentro de soluciones de Internet de las Cosas (IoT-*Internet of Things*) para Ambientes de Vida Asistidos (AAL-*Ambient Assisted Living*), por lo que se ha visto necesaria la elaboración de una revisión sistemática de estudios primarios; mediante la cual se consulta, extrae y recopila la información relevante sobre el tema de interés para el desarrollo del aporte científico del presente trabajo de titulación.

Lo que busca la revisión sistemática planteada es identificar cómo se desarrollan las diferentes aplicaciones o soluciones informáticas de IoT para AAL, como ser construyen sus servicios y como se liberan en la aplicación de Arquitecturas como Microservicios o Arquitecturas Orientadas a Servicios e identificar cual es el proceso que siguen los Desarrolladores, Arquitectos e Ingenieros de software para crear dicha solución.

B. Formulación de las preguntas y sub-preguntas investigación

Se plantea la siguiente pregunta, como objetivo principal de la revisión sistemática efectuada:

¿Cómo se gestionan los servicios en soluciones de IoT para AAL y cuáles son las características de Ingeniería de software tomadas en cuenta para el proceso de desarrollo?

El objetivo de la pregunta es obtener información relevante acerca de las características y procesos de Ingeniería de Software referidos a la implementación, despliegue y mantenibilidad de servicios en soluciones informáticas de IoT para AAL, dado que esta información proporcionará la base para el planteamiento de una correcta metodología que se adapte y mejore en gran magnitud los procesos actuales. Para tener una perspectiva amplia y poder abarcar los estudios primarios más representativos se plantean sub-preguntas de investigación presetadas en la Tabla 3.1.

Sub-pregunta de Investigación	Motivación
RQ1: ¿Cuáles son los tipos de soluciones propuestas para sistemas de IoT para AAL existentes?	El objetivo es conocer cuál es la tendencia en cuanto al surgimiento de aportes en el área temática abordada.
RQ2: ¿Qué aspectos de Ingeniería de Software se consideran al momento de desarrollar aplicaciones empleando microservicios en IoT y AAL?	El objetivo es conocer qué arquitecturas, metodologías y técnicas de Ingeniería de Software son consideradas actualmente y sus tendencias a través de los años.
RQ3: ¿Cuáles son las tecnologías utilizadas en la creación de aplicaciones para soluciones de IoT en AAL?	El objetivo es tener una idea de las herramientas, protocolos y estándares más comunes en las aplicaciones creadas.
RQ4: ¿Qué necesidades o problemas son abordados a través de las soluciones desarrolladas?	El objetivo es saber qué áreas de la salud reciben más atención con un enfoque de ambientes de vida asistidos.
RQ5: ¿Cómo se está desarrollando actualmente la investigación de esta área?	El objetivo es conocer el enfoque de las soluciones planteadas y la relevancia del aporte estudiado.

Tabla 3.1: Sub-preguntas de investigación

C. Protocolo de búsqueda

Con el objetivo de obtener la mayor cantidad de estudios relevantes, se consideran búsquedas automáticas en diferentes bibliotecas digitales de grandes editoriales y se consideran también búsquedas manuales en conferencias de categorías A, B y C.

I. Búsquedas Automáticas

Para el proceso de búsquedas automáticas se han considerado las siguientes bibliotecas digitales, seleccionadas en base a una búsqueda preliminar que arrojó los mejores resultados en las mismas.

- ACM Digital Library
- IEEE Xplore
- Springer Link
- ScienceDirect

Para este proceso se hace uso de una cadena de búsqueda establecida para todas las bibliotecas digitales. Considerando la búsqueda únicamente sobre metadatos, es decir; título del documento, palabras clave y *abstract*.

La determinación de la cadena de búsqueda se basó en el conocimiento previo sobre: AAL, IoT, Microservicios, y en las pruebas de búsqueda, usando diferentes combinaciones de dichos términos. De este modo se seleccionó la combinación de términos que tuvo los mejores resultados posibles en función del objetivo de la investigación planteada. El conjunto de palabras usadas para la selección de estudios primarios se muestra en la Tabla 3.2.

Concepto	**Substring**	**Conector**	**Términos alternativos**
Ambient Assisted Living	assist*	OR	Incluye: *assisted*, *assistance*, *ambient assisted living*
Ambient Assisted Living	AAL	OR	
eHealth	eHealth, e-health	AND	
Internet of Everything	"internet of everything"	OR	
Internet of Things	"internet of things"	OR	
Internet of Everything	IoE	OR	
Internet of Things	IoT	AND	
Microservicios	microservice*	OR	Incluye: *microservice*, *microservices*
Web Services	"web service"	OR	incluye: web services
Service-Oriented Architecture	SOA	OR	
Service-Oriented Architecture	"Service-Oriented Architecture"		
Cadena de búsqueda	***((assist* or AAL or aal or "ambient assisted" or e-health or eHealth) and (ïnternet of everything.ºr ïnternet of things.ºr IoE or IoT) and (microservice* or "web service.ºr SOA or "Service-Oriented Architecture"))***		

Tabla 3.2: Cadena de búsqueda.

II. **Búsquedas Manuales**

En las búsquedas manuales se toman en cuenta conferencias que fueron previamente seleccionadas a través de una búsqueda de los términos: service, web service, Service Oriented, IoT, eHealth, Assisted, en el Portal de Rankings de Conferencias Computing Research and Education Association of Australasia *(http://www.core.edu.au)*. Las conferencias consideradas para la búsqueda manual, calificadas por su título y una breve revisión de su tabla de contenidos, son las siguientes:

- *IEEE International Conference on Service-Oriented Computing (SOCA)*
- *EAI International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (Qshine)*
- *IEEE International Conference on Services Computing (SCC)*
- *International Conference on Service-Oriented Computing (ICSOC)*
- *IEEE International Conference on Web Services (ICWS)*
- *IEEE International Conference on E-health Networking, Application & Services (HealthCom)*

III. **Período de búsqueda**

Se establece el periodo (2009 - 2018) para la búsqueda y selección de estudios primarios, debido al resurgimiento de IoT en 2009 de acuerdo con Ashton et al. (2009) y si bien como se vio en la sección 3.2.1.A el surgimiento del término microservicios se dio en el año 2014, tenemos en cuenta que la Arquitectura de Microservicios es una evolución de la Arquitectura Orientada a Servicios, además de que antes del surgimiento del término ya se aplicaban técnicas propias de Microservicios como lo describe Brown (2016).

IV. **Planificación de selección de estudios primarios** Una vez ejecutadas tanto la búsqueda manual como la automática, los investigadores proceden a evaluar cada estudio en los metadatos (título, *abstract* y palabras clave). De esta preselección resultante se seleccionan aquellos estudios que cumplan con los criterios de inclusión y se excluyen de la revisión sistemática aquellos que cumplan con los criterios de exclusión planteados a continuación.

- **Criterios de inclusión**
  - Estudios que presenten arquitecturas o metodologías basadas en servicios y que estén involucrados con soluciones de IoT para Ambientes de Vida Asistidos.

- Estudios que contemplen aplicaciones, sistemas o plataformas, que involucran el funcionamiento e implementación de microservicios o servicios web haciendo uso de tecnologías de IoT.
- Estudios que implementan arquitecturas de microservicios o SOA para aplicaciones orientadas a ambientes de vida asistidos.

- Criterios de exclusión
  - Artículos introductorios de ediciones especiales: revistas, libros, etc.
  - Estudios duplicados encontrados en las diferentes bibliotecas digitales.
  - Artículos escritos en lenguas diferentes al inglés o al español.
  - Documentos que no tienen sustentos teóricos válidos para respaldar el diseño / metodología / arquitectura propuesta.

v. **Criterios de extracción de datos**

Para responder las sub-preguntas de investigación planteadas, se definen criterios de extracción de información, los cuales se muestran a continuación. La importancia de definir los criterios de extracción radica en evitar el sesgo de los investigadores para que sus expectativas no influyan en los análisis de los estudios.

RQ1: ¿Cuáles son los tipos de soluciones propuestas para sistemas de IoT para AAL existentes?

Los criterios de extracción para RQ1 se muestran en la Tabla 3.3.

EC1 Solución planteada	EC2 Ambiente de despliegue	EC3 Modelo de Servicio
• Metodología	• Web	• Infraestructura como servicio (IaaS)
• Arquitectura	• Aplicación móvil	• Plataforma como servicio (PaaS)
• Framework	• Escritorio	• Software como servicio (SaaS)
• Prototipo	• Aplicación embebida	• IoT como servicio (IoTaas)
• Aplicación de software	• No especifica	• Context as a Service (CaaS)
• Dispositivo de hardware		• Middleware as a Service (MWaaS)
• Otros		• Otros

Tabla 3.3: Criterios de Extracción QR1

RQ2: ¿Qué aspectos de Ingeniería de Software se consideran al momento de desarrollar aplicaciones empleando microservicios en IoT y AAL?

Los criterios de extracción para RQ2 se muestran en la Tabla 3.4.

EC4 Tipo de Arquitectura de software	EC5 Plataforma de despliegue	EC6 Modelo de Despliegue Cloud
• Modelo Vista Controlador (MVC). • Cliente-servidor • Arquitectura de Microservicios (MSA). • Arquitectura publish/-subscribe OSGi • Arquitectura orientada a servicios (SOA) • Pipeline • Arquitectura dirigida por modelos (MDA) • La arquitectura basada en eventos (EDA) • Arquitectura Propia • Otras	• Hosting • Cloud computing • Fog computing • Edge computing • Localmente • No especifica	• Pública • Privada • Híbrida • No especifica
EC7 Fases de Entrega Continua de software que se consideran	**EC8 Metodología de desarrollo de software Aplicada**	**EC9 Aspectos de calidad según ISO 25010**
• Integración Continua • Pruebas • Producción • No Aplica	• Ágiles • Cascada • Evolutivas • DevOps • No Especifica	• Eficiencia • Usabilidad • Fiabilidad • Disponibilidad • Seguridad • Mantenibilidad • Interoperabilidad • Otros

Tabla 3.4: Criterios de Extracción QR2

RQ3: ¿Cuáles son las tecnologías utilizadas en la creación de aplicaciones para soluciones de IoT en AAL?

Los criterios de extracción para RQ3 se muestran en la Tabla 3.5.

EC10 Bases de datos	EC11 Formato de intercambio de datos	EC12 Estándares de comunicación
• Relacional • No relacional • No especifica	• XML • JSON • HTML	• Z-wave • KNX • UPnP

	• No especifica	• JMS • No especifica
EC13 Conjuntos de estándares y sistemas de conceptos orientados a la salud.	**EC14 Tipo de servicio web proporcionado.**	**EC15 Protocolos de comunicación (modelo TCP-IP)**
• HL7 *Reference Information Model* • HL7 *Electronic Health Record* • X73 • ISO 13940 (ContSys) • PQRST • No especifica	• REST • SOAP-WS • No especifica	• XMPP • MQTT • CoAP • AMQP • HTTPS • No especifica
EC16 Especifica herramientas		
	• Si • No	
EC17 Herramientas de Gestión de Contenedores	**EC18 Middlewares**	**EC19 Herramientas de Desarrollo de SW**
• Docker • Knopflerfish	• RabbitMQ • VIRTUS • OpenHAB • Hydra Middleware	• Java EE • Apache Struts • Node.js
EC20 Servidores de Aplicaciones	**EC21 Frameworks**	**EC22 Herramientas Cloud Management**
• Glassfish server • Apache Tomcat	• Spring • Ruby on Rails • Django • ASP.NET Apache Felix	• OpenStack • Microsoft Azure IoT Hub • IBM Bluemix Google App Engine • Amazon Web Services
EC23 Herramientas de Gestión de datos	**EC24 Herramientas estándar**	**EC25 Lenguaje de programación (Implementación de servicios)**
• RoQua • IBM Watson	• Web Processing Service(WPS) • Web Feature Service (WFS)	• C++ • Javascript

• Websphere Lombardi Edition • Websphere ILOG rules • Apache Kafka • Hibernate	• Web Socket • DPWS	• PHP • Java • R • Python Web • No especifica

Tabla 3.5: Criterios de Extracción RQ3

RQ4: ¿Qué necesidades o problemas son abordados a través de las soluciones desarrolladas?

Los criterios de extracción para RQ4 se muestran en la Tabla 3.6.

EC26 Orientación	EC27 Tipo de servicio de salud	EC28 Dominio de cuidado de la salud identificado
• Pacientes • Cuidadores • Familiares • Profesionales de la Salud • Administradores de eHealth system • Sistemas con Inteligencia Artificial • Otros	• Asistencia para medicación • Asistencia para personas mayores • Asistencia de estilo de vida • Manejo de enfermedades crónicas • Cuidados paliativos • Gestión de datos de salud • Gestión de salud personal • Servicios de rehabilitación • Monitoreo de la salud	• Home Health • e-Health • M-Health (mobile health) • Ubiquitous Health • Hospital Management. • WSN (Wireless sensor networks)

Tabla 3.6: Criterios de Extracción RQ4

RQ5: ¿Cómo se está desarrollando actualmente la investigación de esta área?

Los criterios de extracción para RQ5 se muestran en la Tabla 3.7.

EC29 Campos de aplicación de los estudios	EC30 Métodos de validación	EC31 Tipo de estudio
• Industria • Académico	• Experimentos controlados • Cuasiexperimentos • Pruebas de conceptos • Casos de estudio	• Nuevo • Extensión

Tabla 3.7: Criterios de Extracción RQ5

3.3. Ejecución de la Revisión

Esta fase consiste en establecer las pautas para seleccionar los estudios relevantes de manera adecuada, evaluarlos por su calidad. De esta manera esta etapa consiste de tres pasos: (i) selección de estudios primarios, (ii) evaluación de calidad y (iii) método de análisis y síntesis.

3.3.1. Selección de Estudios Primarios

Como resultado de la ejecución de las búsqueda automática se obtuvieron 1502 estudios y como resultado de la ejecución de las búsquedas manuales se obtuvieron 36 estudios, danto un total de 1538 estudios encontrados mediante las búsquedas, como se lo indica en la Tabla 3.8.

Búsqueda Automática	
Biblioteca Digital	**Nro. de estudios encontrados**
ACM	17
IEEE	28
Science Direct	99
Springer Link	1364
Subtotal	1508
Búsqueda Manual	
Conferencia	**Nro. de estudios encontrados**
HealthCom	22
ICSOC	2
ICWS	1
SCC	3
SOCA	4
Qshine	4
Subtotal	**36**
Total	**1538**

Tabla 3.8: Resultados de la búsqueda automática y búsqueda manual

Dentro de los 1502 estudios resultantes de la búsqueda automática se realiza un pre-selección de estudios primarios, revisando manualmente los metadatos (*abstract*, palabras clave y título del documento). Obteniendo así 52 documentos preseleccionados. Como se muestra a continuación en la Tabla 3.9.

Búsqueda Automática (Pre-selección)	
Biblioteca Digital	**Num de estudios preseleccionados.**
ACM	11
IEEE	8
Science Direct	14
Springer Link	19
Total preseleccionados (búsqueda automática)	**52**

Tabla 3.9: Resultados del proceso de preselección en la búsqueda automática

Con un total de 88 estudios preseleccionados (búsqueda manual + búsqueda automática) se efectúa el siguiente paso del protocolo de revisión planteado en la sección 3.1 de este documento. Se evalúa cada estudio para determinar si debe o no ser incluido. Cualquier discrepancia en este proceso se resuelve en

consenso al examinar completamente el estudio en conflicto. Se han incluido los estudios que cumplan por lo menos uno de los criterios de inclusión, siempre y cuando no cumplan ninguno de los criterios de exclusión. En la Tabla 3.10 se indica un análisis cuantitativo de los 48 estudios primarios que se obtuvieron como resultado de la aplicación de criterios de inclusión y exclusión. La cuarta columna de la Tabla 3.10 indica el porcentaje de los 48 estudios primarios que representa cada elemento de la fila de estudios incluidos en el total de la selección de estudios primarios, permitiendo notar el hecho de que se ha obtenido un 56,25 % de estudios primarios resultante de las búsquedas automáticas y un 43.75 % resultante de las búsquedas manuales, lo que demuestra la importancia de realizar estos dos tipos de búsqueda en la revisión sistemática.

Estudios Primarios seleccionados de la Búsqueda Automática			**Porcentaje Estudios Incluidos**
Biblioteca Digital	**Estudios Excluidos**	**Estudios Incluidos**	**Estudios Primarios**
ACM	10	7	**14,58 %**
IEEE	23	5	**10,42 %**
Science Direct	92	7	**14,58 %**
Springer Link	1356	8	**16,67 %**
Subtotal	**1475**	**27**	**56,25 %**
Estudios Primarios seleccionados de la Búsqueda Manual			**Porcentaje Estudios Incluidos**
Conferencia	**Estudios Excluidos**	**Estudios Incluidos**	**Estudios Primarios**
HealthCom	6	16	**33,33 %**
ICSOC	1	1	**2,08 %**
ICWS	1	0	**0,00 %**
SCC	2	1	**2,08 %**
SOCA	2	2	**4,17 %**
Qshine	3	1	**2,08 %**
Subtotal	**15**	**21**	**43,75 %**
Total	**1490**	**48**	**100 %**

Tabla 3.10: Resultados de la búsqueda automática y búsqueda manual

Del total de 1538 estudios analizados, 1490 han sido rechazados o excluidos, mientras que 48 han sido aceptados o incluidos en la revisión sistemática luego de aplicar el protocolo de búsqueda planteado en la sección 3.2.1.C. En la Figura 3.1 se muestra un resumen de los estudios analizados mediante la relación

de estudios incluidos y excluidos en cuanto al total de estudios analizados, en una escala logarítmica para lograr una mejor presentación de los datos. Los estudios incluidos en la revisión sistemática se pueden consultar en el Anexo A.1

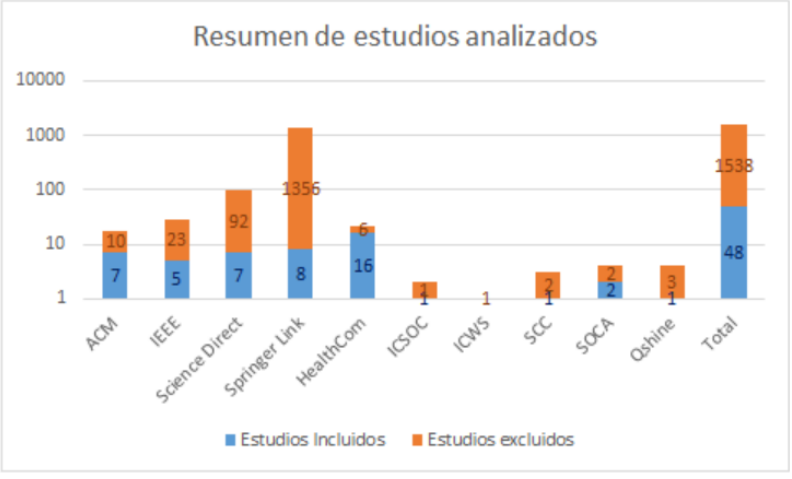


Figura 3.1: Resumen de estudios incluidos y excluidos.

3.3.2. Evaluación de la calidad

Además de los criterios de inclusión y exclusión, es importante llevar a cabo una evaluación de calidad de los estudios primarios incluidos en la revisión sistemática. Para este propósito, se usa una escala de Likert de tres puntos, basada en el número de citas de cada estudio. La Tabla 3.11 muestra los resultados de la evaluación de acuerdo con los criterios de calificación: (-1) el estudio no tiene citas, (0) el estudio tiene de una a tres citas, (+1) el estudio tiene más de tres citas. El recuento de citas se hizo tomando en cuenta las citas no repetidas que muestran el resultado de la búsqueda de cada paper en Google Scholar, además se omitió los estudios de tipo *Chapter* (3 estudios no evaluados), debido a que sus citaciones no suelen ser contabilizadas.

Descripción	Puntaje	Cantidad	Porcentaje
No tiene citas	-1	9	**20,00 %**
(1 a 3) citas	0	14	**31,11 %**
Más de 3 citas	1	22	**48,89 %**

Tabla 3.11: Evaluación de la calidad de los estudios primarios

A pesar de tener 9 estudios primarios que no han tenido cita hasta la fecha se establece que la calidad es aceptable debido a que 6 de los 9 estudios han sido publicados en el año 2017 o 2018, por lo que se justifica su falta de citas literarias.

3.3.3. Métodos de análisis y síntesis

Para la evaluación de los estudios se aplica un método de análisis cualitativo y cuantitativo para luego sintetizar los resultados cuantitativamente, basado en tres pasos:

A. Responder a los 31 criterios de extracción de datos por cada uno de los 48 estudios primarios seleccionados.

B. Contabilizar los resultados obtenidos en cada uno de los criterios de extracción de datos por cada pregunta de investigación.

C. Elaboración de histogramas que muestran una representación de distribuciones de frecuencias de los criterios más relevantes que han sido evaluados durante la revisión, se usan también diagramas de burbujas para representar gráficamente las relaciones existentes entre las sub-preguntas de investigación más relevantes.

En los resultados obtenidos (revisar Anexo A.2) el porcentaje total excede el 100 % en diversos criterios debido a que los mismo son de carácter excluyente, es decir; un mismo estudio puede adoptar varias alternativas como respuesta a un criterio de extracción de datos. En base a ello, se puede realizar las siguientes observaciones:

En la Figura 3.2 se puede apreciar los diferentes tipos de soluciones encontradas en la revisión sistemática, evidentemente el principal tipo de solución planteada ofrecido es una aplicación de software, sin embargo; también se puede observar la poca existencia de trabajos relacionados con la creación de metodologías, es por ello que la elaboración de una metodología es necesaria.

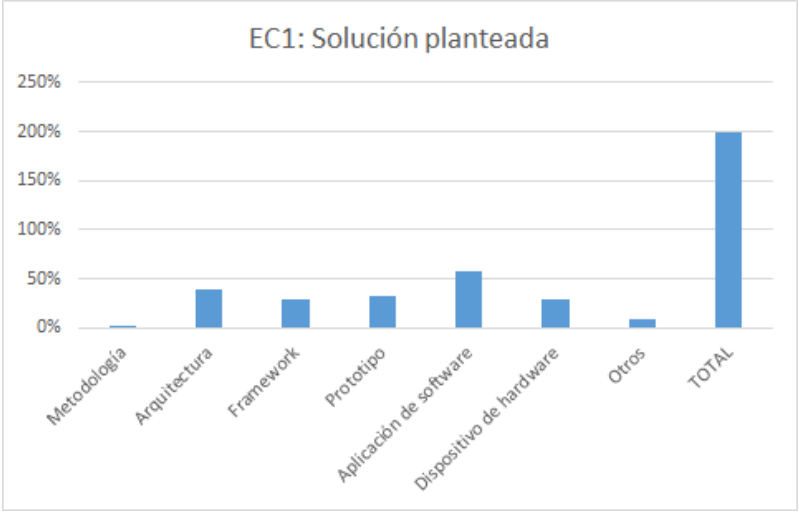


Figura 3.2: Porcentaje de estudios correspondientes a EC1: Solución planteada .

En la Figura 3.3 se puede observar que el tipo de arquitectura de software principalmente adoptado es de tipo SOA, seguido por la arquitectura Cliente-servidor y la arquitectura de microservicios (MSA).

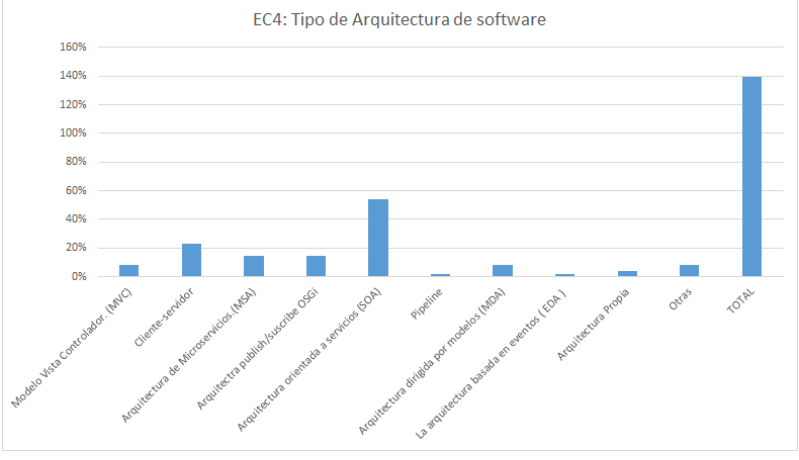


Figura 3.3: Porcentaje de estudios correspondientes a EC4: Tipo de Arquitectura de software.

Respecto a los aspectos tecnológicos la Figura 3.4 muestra que en sólo el 60 % de los estudios se especifica las herramientas utilizadas, además; se encontró que la mayoría de aplicaciones no utilizan estándares y conceptos relacionados a la salud, por otro lado; entre los protocolos de comunicación utilizados el predominante es el protocolo HTTPS.

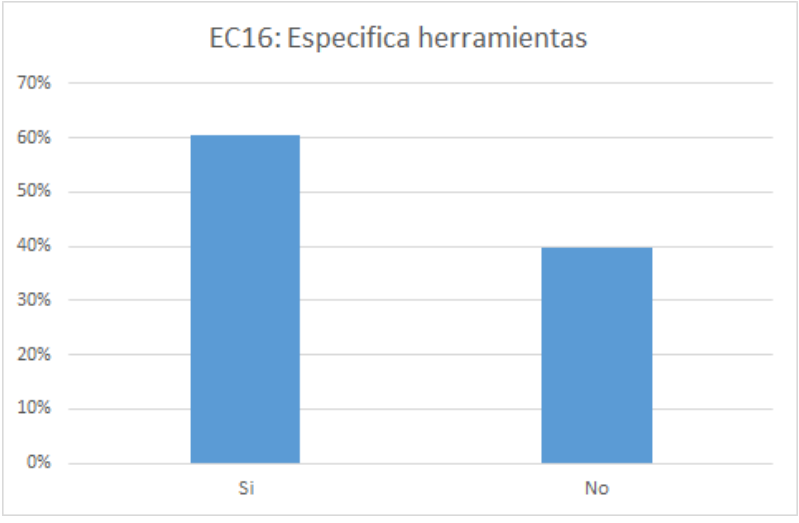


Figura 3.4: Porcentaje de estudios correspondientes a EC16: Especifica herramientas

Entre las necesidades principales identificadas en la Figura 3.5 se ve que las aplicaciones son orientadas principalmente a pacientes, cuidadores y profesionales de la salud, mientras que la Figura 3.6 muestra que el principal servicio brindado por las aplicaciones es el monitoreo de salud.

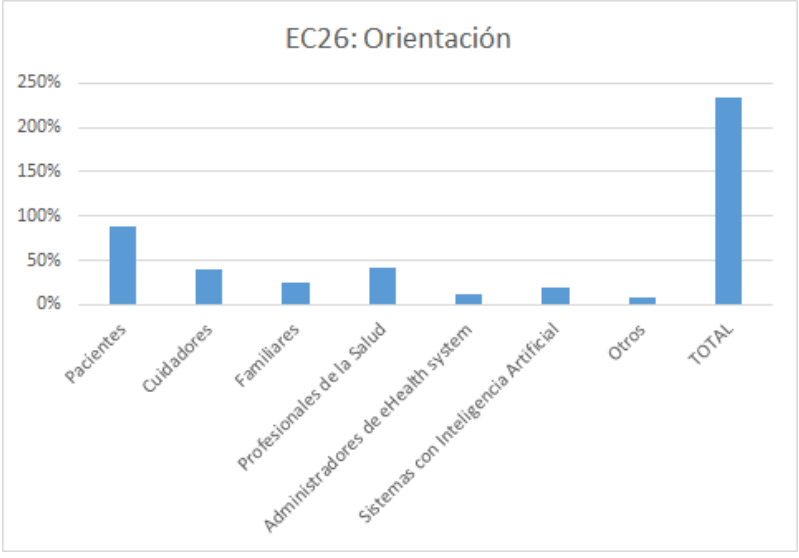


Figura 3.5: Porcentaje de estudios correspondientes a EC26: Orientación

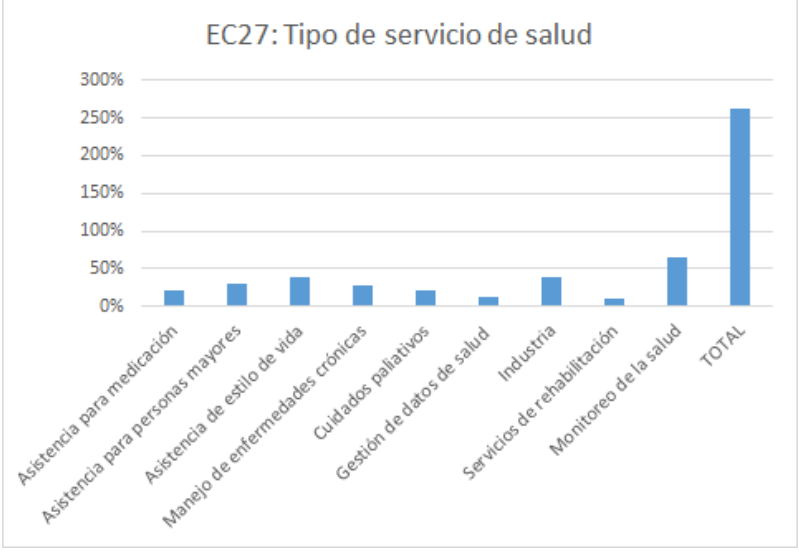


Figura 3.6: Porcentaje de estudios correspondientes a EC27: Tipo de servicio de salud.

Finalmente, la Figura 3.7 muestra el cómo se están abordando los estudios, siendo la mayoría de carácter académico.

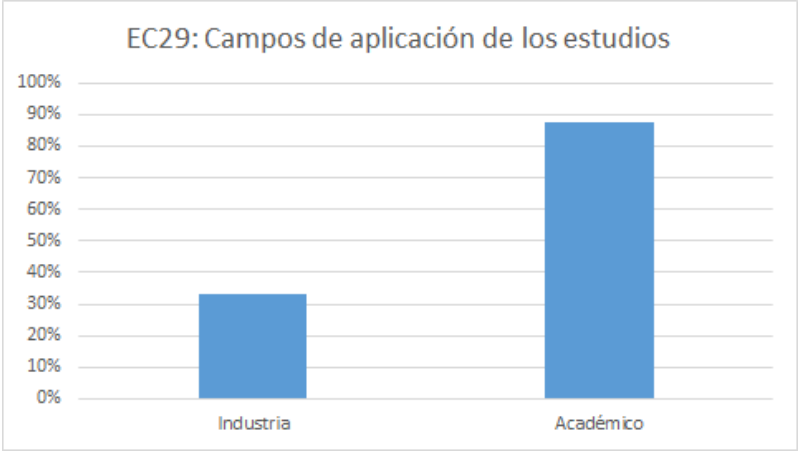


Figura 3.7: Porcentaje de estudios correspondientes a EC29: Campos de aplicación de los estudios.

Para la obtención de resultados más profundos, se realizó diagramas de dispersión entre los criterios de extracción. La Figura 3.8 muestra en el eje de las abscisas a los criterios EC16 - Especifica Herramientas y EC17 - Campos de aplicación de los estudios, mientras que en el eje de las ordenadas se presenta al criterio EC1: Solución planteada. De ello se puede observar que la mayoría de soluciones planteadas en el campo académico son de tipo prototipo, y que además conforman en mayor tipo de solución que especifica herramientas. Por otra parte, la metodología conforma el menor tipo de soluciones encontradas.

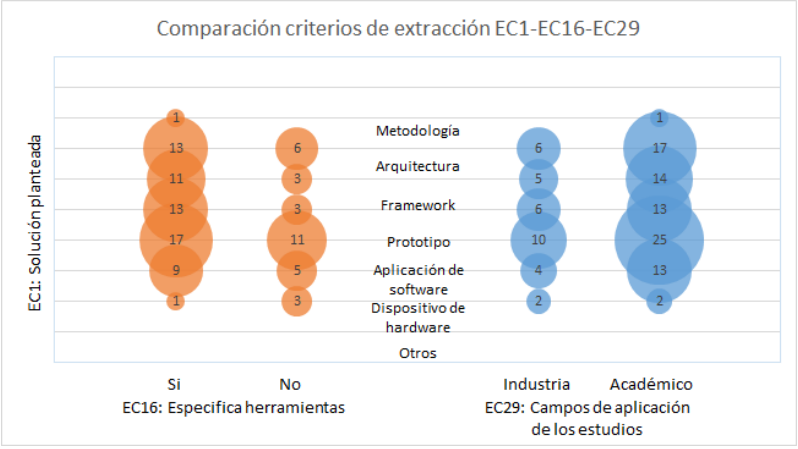


Figura 3.8: Comparación entre EC1: Solución planteada, EC 16: Especifica herramientas y EC29: Campos de aplicación de los estudios.

En la Figura 3.9 se muestra en el orden de las abscisas el tipo de arquitectura mientras que en el eje de las ordenadas se muestra el tipo de solución planteada, el tipo de arquitectura más utilizado corresponde a la arquitectura SOA aplicado en aplicaciones de software y en la creación de arquitecturas. Con respecto a las metodologías propuestas, las arquitecturas empleadas son la arquitectura OSGI y la arquitectura SOA, por lo tanto; el aporte de este trabajo de titulación, que es la elaboración de una metodología basada en microservicios conformaría un aporte nuevo para la comunidad científica.

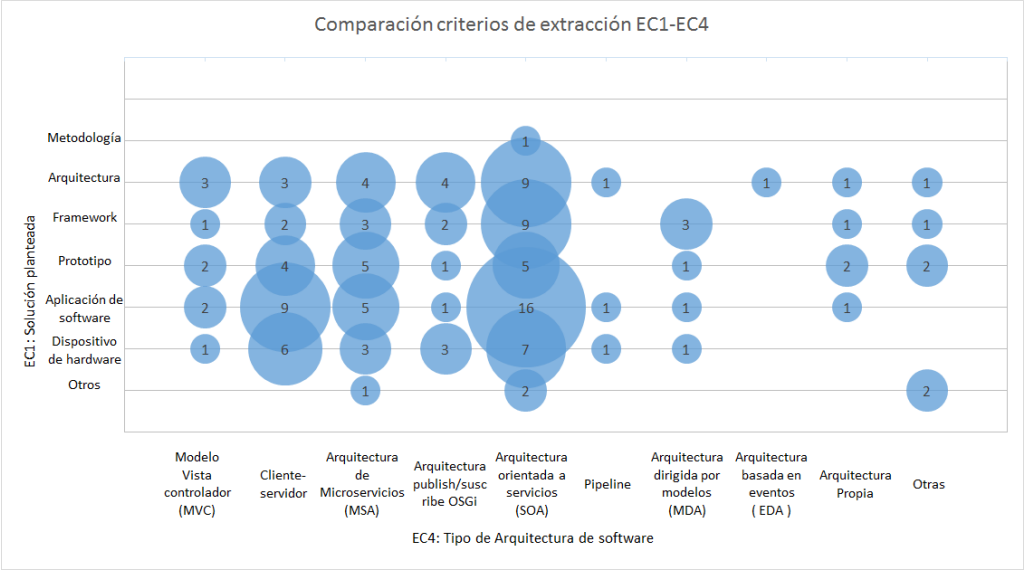


Figura 3.9: Comparación entre EC1: Solución planteada y EC4: Tipo de Arquitectura de software.

La Figura 3.10 muestra una comparación entre los tipos de servicios de salud y el tipo de persona hacia quien va orientado la solución, entre los hallazgos más relevantes se tiene que los pacientes son los principales beneficiados, siendo asistidos adultos mayores, asistencia en estilo de vida, gestión de salud personal, y monitoreo de la salud. De igual manera los profesionales de la salud se benefician al realizar tareas de monitoreo de la salud.

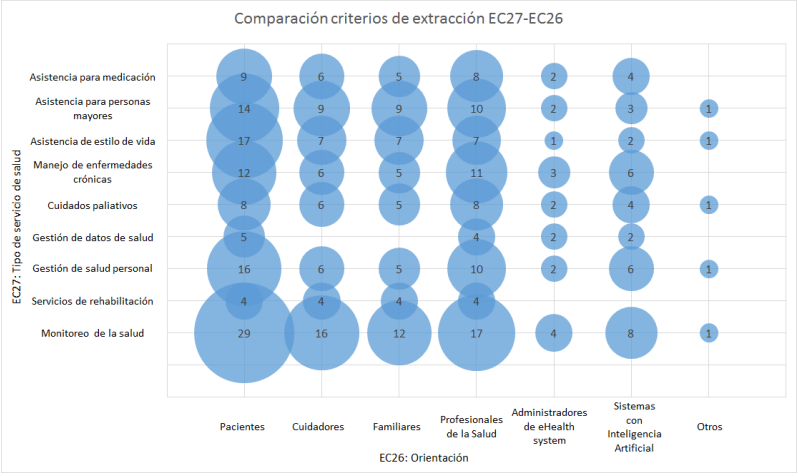


Figura 3.10: Comparación entre EC27: Tipo de servicio de salud y EC26: Orientación.

3.4. Difusión de Resultados

La revisión sistemática inició con un total de 1538 estudios obtenidos a partir de una revisión sistemática y manual, de estos se descartaron 1450 estudios en una revisión de su *abstract*, palabras clave y contenido general, los 88 estudios preseleccionados pasaron a una evaluación de criterios de inclusión y exclusión, de los cuales finalmente se obtuvieron 48 estudios primarios, estos se representan en la Figura 3.11 en una línea de tiempo, como se observa; la mayoría de los estudios seleccionados corresponden a los años 2015 y 2017.

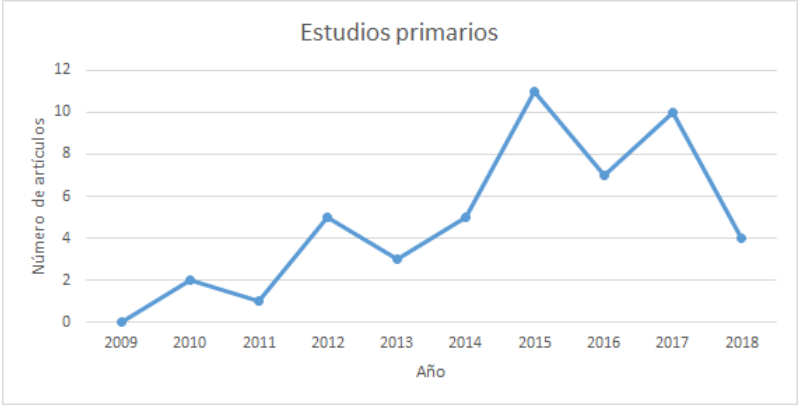


Figura 3.11: Relación estudios primarios - Año de publicación.

En la Figura 3.12 presenta del criterio de extracción 4: Tipo de Arquitecturas de Software las arquitecturas más encontradas en los estudios seleccionados y su trascendencia a través de los años. Como se aprecia, entre los años 2015 y 2017 las arquitecturas Cliente-servidor, SOA, y OSGi presentan una menor concurrencia, mientras que la Arquitectura de Microservicios revela una mayor trascendencia; lo cual sugiere la orientación del desarrollo de aplicaciones de IoT para AAL hacia ese tipo de arquitectura.

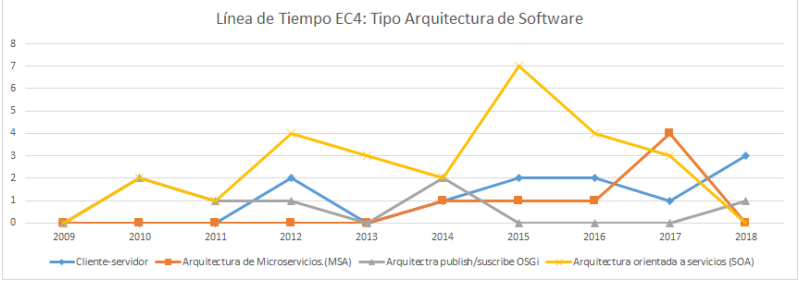


Figura 3.12: Trascendencia de Tipos de Arquitectura de Software.

A continuación, se presentan los resultados obtenidos para cada sub-pregunta de investigación, recordando que estas preguntas responden a la pregunta siguiente que fue el objetivo principal de esta revisión.

¿Cómo se gestionan los servicios en soluciones de IoT para AAL y cuáles son las características de Ingeniería de software tomadas en cuenta para el proceso de desarrollo?

RQ1: ¿Cuáles son los tipos de soluciones propuestas para sistemas de IoT para AAL existentes?

- De los 48 estudios seleccionados para la revisión sistemática se encontró que un 58 % de soluciones son aplicaciones de software, mientras que sólo un 2 % se centran en la creación de metodologías. Los estudios primarios se enfocan en su mayoría en aplicaciones de entorno web y aplicaciones móviles, además; un 60 % implementan un modelo de servicio (SaaS). Estos resultados se deben a que la mayoría de aplicaciones de software desarrolladas ofrecen una solución para solventar necesidades particulares como monitoreo de la salud y asistencia de estilo de vida.

RQ2: ¿Qué aspectos de Ingeniería de Software se consideran al momento de desarrollar aplicaciones empleando microservicios en IoT y AAL?

- El tipo de arquitectura más empleado corresponde a la Arquitectura Orientada a Servicios (SOA) debido a la aceptación obtenida a través de los años, sin embargo; como muestra la Figura 3.12, a partir de 2015 esta arquitectura ha comenzado a ser menos implementada, por otra parte; se muestra un aumento de investigación sobre la arquitectura de microservicios (MSA) debido al mayor grado de independencia que ofrece.

- *Cloud computing* corresponde a la plataforma de despliegue más utilizada, sin embargo; donde sólo el 21 % de los estudios indican el modelo de despliegue utilizado. Respecto a las fases de entrega continua de software, un 56 % considera este criterio siendo en su mayoría en fases de prueba.

- De los estudios analizados se encontró que sólo el 40 % especificaba la metodología de desarrollo de software utilizada.

- De los aspectos de calidad mencionados en la ISO 25010 los más considerados por los estudios son seguridad, usabilidad, interoperabilidad con 25, 20, 17 estudios respectivamente.

RQ3: ¿Cuáles son las tecnologías utilizadas en la creación de aplicaciones para soluciones de IoT en AAL?

- Respecto a las tecnologías implementadas, un 50 % de los trabajos especifican que implementan bases de datos relacionales y no relacionales.

- El 56 % de los estudios especifican el formato de salida empleado, siendo XML el formato más utilizado.

- El 94 % de los estudios no establece un estándar de comunicación.

- Al considerar que las aplicaciones que los estudios seleccionados se orientan a la salud se consideró estándares y conjuntos de conceptos orientados a dicho campo, sin embargo; solo un 15 % de los estudios implementan estándares tales como: *HL7 Electronic Health Record* o el *HL7 Reference Information Model*.

- El tipo de servicio web predominante es el servicio REST ya que se encontró en 20 estudios. Mientras que el principal protocolo de comunicación es el protocolo HTTPS encontrado en un 30 % de los estudios.

- El 60 % de los estudios indicó las herramientas que implementa siendo de las cuales, las más destacadas son: Java, Javascript, Node.js, Apache Tomcat, Amazon Web Services Platform, Microsoft Azure IoT Hub y Web Processing Service (WPS).

RQ4: ¿Qué necesidades o problemas son abordados a través de las soluciones desarrolladas?

- Las aplicaciones de la salud están orientadas a diferentes usuarios, es por ello que se encontraron resultados solapados. Dicho ello, el 88 %, 42 % y 40 % de las aplicaciones son orientadas para cuidadores, profesionales de la salud y cuidadores respectivamente.

- De igual manera, se encontró aplicaciones que se orientan a diferentes tipos de servicios de salud, siendo las principales el monitoreo de la salud, la industria, y la asistencia a personas mayores.

- *Home Health*, WSN (*Wireless Sensor Networks*) y M-Health (*Health mobile*) son los principales dominios del cuidado de la salud que se encontraron, siendo encontradas en el 56 %, 44 % y 40 % de las publicaciones respectivamente. Es por ello que estos dominios deben ser considerados en el desarrollo de futuras aplicaciones.

RQ5: ¿Cómo se está desarrollando actualmente la investigación de esta área?

- Se encontró que la investigación en esta área es en su mayoría nueva ya que el 67 % de los estudios seleccionados son nuevos. Sin embargo el 88 % de los estudios son académicos. Además de ello, el 58 % de los estudios son validados a través de casos de estudio.

Edwin Cabrera Alvarado
Paola Cárdenas Cárdenas

Capítulo 4

Metodología

En este capítulo se presenta MicroIoT (Metodología basada en Microservicios para Internet de las Cosas). Se aborda la contribución que representa este trabajo de titulación, con el diseño de una metodología que facilite la creación de aplicaciones de Internet de las Cosas para Ambientes de Vida Asistidos, aplicando microservicios, para ello; la metodología propone las actividades necesarias para el análisis, diseño, desarrollo y mantenimiento continuo para soluciones de Internet de las Cosas en Ambientes de Vida Asistidos. Este capítulo presenta: (i) el contexto en el que se desenvuelve la metodología propuesta, (ii) la identificación de la necesidad de crear dicha metodología, (ii) las bases conceptuales de Ingeniería de Software sobre las cuales se planteó la metodología, y (iv) MicroIoT la metodología propuesta.

4.1. Contexto

MicroIoT representa una metodología que permite la creación de aplicaciones basadas en microservicios que a más de ser aplicada a los dominios de IoT y AAL, puede adaptarse a otros de manera muy sencilla. En este trabajo se ha propuesto el uso de microservicios, debido a las características propias de los mismos (resiliencia, disponibilidad, tolerancia a fallos, fácil escalabilidad), lo que concuerda plenamente con las necesidades de IoT para AAL. Además, una motivación destacable ha sido la de ofrecer un aporte que beneficie a los sectores vulnerables de la sociedad.

4.2. Enfoque a Microservicios

El desarrollo tradicional de aplicaciones siempre ha seguido un enfoque monolítico, que si bien se divide en artefactos y funcionalidades (clases, métodos), toda la lógica de negocio está altamente acoplada; lo que significa que un cambio en la aplicación monolítica implica que se reconstruya y vuelva a desplegar toda la aplicación. Además, un sistema monolítico muestra severas limitaciones al considerar la ampliación de los sistemas a múltiples usuarios y los ciclos de despliegue rápido; de ahí, los cambios requieren que toda la aplicación se vuelva a implementar.

Por otra parte, el crecimiento de la tecnología ha hecho que los dispositivos IoT evolucionen y se creen diferentes protocolos de comunicación entre sus nodos, lo que dificulta la conectividad, la escalabilidad y la integración, especialmente en marcos fabricados monolíticamente (Uviase and Kotonya, 2018). El Internet de las Cosas (IoT) se está adoptando en diferentes dominios de aplicaciones (Krylovskiy et al., 2015) y la cantidad de dispositivos conectados está creciendo rápidamente, Gartner predice cerca de 21 mil millones de dispositivos para 2020. Cisco incluso predice cerca de 50 mil millones de dispositivos conectados (Butzin et al., 2016). Dado el gran número de tecnologías que se involucran en el desarrollo de una aplicación de Internet de las Cosas, necesarias debido a que prácticamente todos los días se desarrolla una tecnología diferente que deja obsoletas a otras o que éstas deban ser desplegadas en ecosistemas diferentes que evolucionan día a día. Por todos estos motivos, se considera que una aplicación monolítica no es una solución adecuada para el dominio tratado en este trabajo de titulación.

Entonces, debido a la necesidad de soportar un gran número de usuarios y un procesamiento de datos significativo, Internet de las Cosas requiere un enfoque específico para el problema de proporcionar suficiente escalabilidad y rendimiento, apuntando claramente a la distribución del esfuerzo entre un gran número de servicios pequeños y especializados (Vresk and Čavrak, 2016).

Por otro lado, la Arquitectura Orientada a Servicios (SOA) ofrece un potente marco de trabajo para admitir la conectividad, la interoperabilidad y la integración en sistemas de IoT, y constituye la columna vertebral de los marcos actuales de IoT. Si bien los objetivos de SOA son principalmente para mejorar la interoperabilidad de aplicaciones de IoT, su uso monolítico en los marcos recientes de IoT amplifica aún más el problema de la escalabilidad, especialmente con la enorme cantidad de "cosas" interconectadas. Los sistemas IoT tienden a expandirse y con el tiempo, un marco SOA capaz se vuelve demasiado inamovible para manejar la extensibilidad del sistema. Consecuentemente, una arquitectura de microservicios fragmenta diferentes sistemas de IoT permitiendo atender adecuadamente la evolución y extensibilidad del sistema (Uviase

and Kotonya, 2018).

Basados en la definición de la arquitectura de microservicios vista en el Capítulo 2 de este documento y de acuerdo con Butzin et al. (2016), se evidencia la existencia de algunas similitudes en los objetivos de los microservicios y el Internet de las Cosas:

- Comunicación ligera.
- Software implementable independiente.
- Un mínimo de gestión centralizada.
- Técnicas y tecnologías de desarrollo independientes.

No obstante, una arquitectura de microservicios rompe el esquema de desarrollo de software tradicional y el esquema organizacional, por ello; se considera el desarrollo de software ágil, el cual abarca el proceso desde el inicio, en donde se desarrolla la visión del sistema, se define el alcance del proyecto y se justifica el caso de negocio; hasta la transición en la que se entrega el software al cliente; mientras que el enfoque organizacional DevOps, abarca el proceso desde la planificación hasta las operaciones, y a menudo incluye departamentos como recursos humanos. Los esfuerzos ágiles a menudo terminan en la fase de transición del desarrollo a las operaciones. Las operaciones sobre la entrega de software (es decir, el intervalo en el que el software está disponible para el usuario y es mantenido por las operaciones) está cubierto por DevOps. Por este motivo MicroIoT basa sus cimientos en estos conceptos, para proponer una metodología que abarque el ciclo de vida completo de un sistema de software de IoT para AAL basado en microservicios.

4.3. Bases de la metodología

DevOps es un enfoque organizacional que ha tenido mucho éxito y popularidad en los últimos años, enfatiza la empatía y la colaboración multifuncional de equipos de trabajo, especialmente para desarrollo y las operaciones de TI (Tecnologías de la Información), permitiendo desarrollar software de alta calidad, para operar sistemas resilientes y acelerar la entrega de cambios (Dyck et al., 2015). En una arquitectura de microservicios, cada microservicio es independiente, además puede ser agnóstico de tecnologías y lenguajes de programación, por lo que de acuerdo con Ebert et al. (2016) la Arquitectura de Microservicios necesita de DevOps para una óptima aplicación de la misma y para manejar de mejor manera cada una de las actividades que implica su uso, y así ofrecer a los interesados un producto final de calidad y altamente resiliente.

La mayor ventaja de DevOps es el rápido ciclo de cada una de las entregas al integrar el desarrollo y las operaciones (Ebert et al., 2016). Por ello se consideró el planteamiento de una metodología ágil, considerando que una de las cualidades más destacables en una metodología ágil es su sencillez, tanto en su aprendizaje como en su aplicación (Canós and Letelier, 2012), además se consideran desarrollos incrementales e iterativos, basados en entregas continuas.

En una metodología ágil los requisitos y soluciones evolucionan con el tiempo según la necesidad del proyecto y el trabajo es realizado por equipos autoorganizados y multidisciplinarios, inmersos en un proceso de toma de decisiones a corto plazo. Cada iteración del ciclo de vida incluye: análisis de requisitos, diseño, establecimiento de la arquitectura, validación, desarrollo continuo, evaluación continua, integración continua, despliegue continuo y monitoreo continuo (Figueroa et al., 2008). Teniendo claro que el objetivo de cada iteración no es agregar toda la funcionalidad, sino incrementar el valor del software (Canós and Letelier, 2012).

Como menciona Richardson (2014), un problema puede descomponerse en base a las capacidades de negocio o en base a su dominio, En esta metodología la actividad de diseño de la entrega implementa el estilo de Diseño Dirigido por Dominio (DDD) debido a que es el más usado y recomendado por diversos autores: Wasson (2017), Sharma (2017), Steinegger et al. (2017), de la Torre (2017), Vernon (2013), entre muchos otros. Sin embargo; en trabajos futuros esta actividad puede ser adecuada para que el diseño de entregas pueda realizarse aplicando el Diseño por capacidades de negocio, entre otros.

4.4. Metodología propuesta

MicroIoT (Metodología basada en Microservicios para Internet de las Cosas) es la propuesta del presente trabajo de titulación, se trata de una metodología ágil, alineada con el enfoque organizacional DevOps y basada en un desarrollo iterativo e incremental, en el cual, en cada iteración se obtiene una nueva entrega que agrega funcionalidad y valor al sistema de software.

Los microservicios fragmentan aplicaciones complejas para entregar servicios o software rápidamente, por lo que se acopla adecuadamente a los principios de una metodología ágil, lo cuales se analizaron en el Capítulo 2, además como Ebert et al. (2016) afirma, DevOps proporciona el marco para desarrollar, implementar y administrar el ecosistema de contenedores de microservicios. Por ello MicroIoT se alinea a los procesos que establece DevOps, además ofrece actividades con mayor detalle y solidez basándose en conceptos de Ingeniería de Software. MicroIoT se centra en profundizar la etapa de planificación de DevOps, desglosando la planificación en las actividades: i) análisis de re-

querimientos, ii) diseño de la entrega, iii) adecuación de la arquitectura y iv) validación del diseño y arquitectura de la entrega o solución. Las actividades de desarrollo y pruebas, despliegue y operaciones, están alineadas con sus homónimos de DevOps, esto se debe a que las entregas de calidad con un ciclo de tiempo corto, requieren un alto grado de automatización, con el uso de herramientas y tecnologías (Ebert et al., 2016). La Figura 4.1 muestra el ciclo de vida completo que seguirá cada entrega (ya sea única o varias) y su relación con la arquitectura de referencia DevOps.

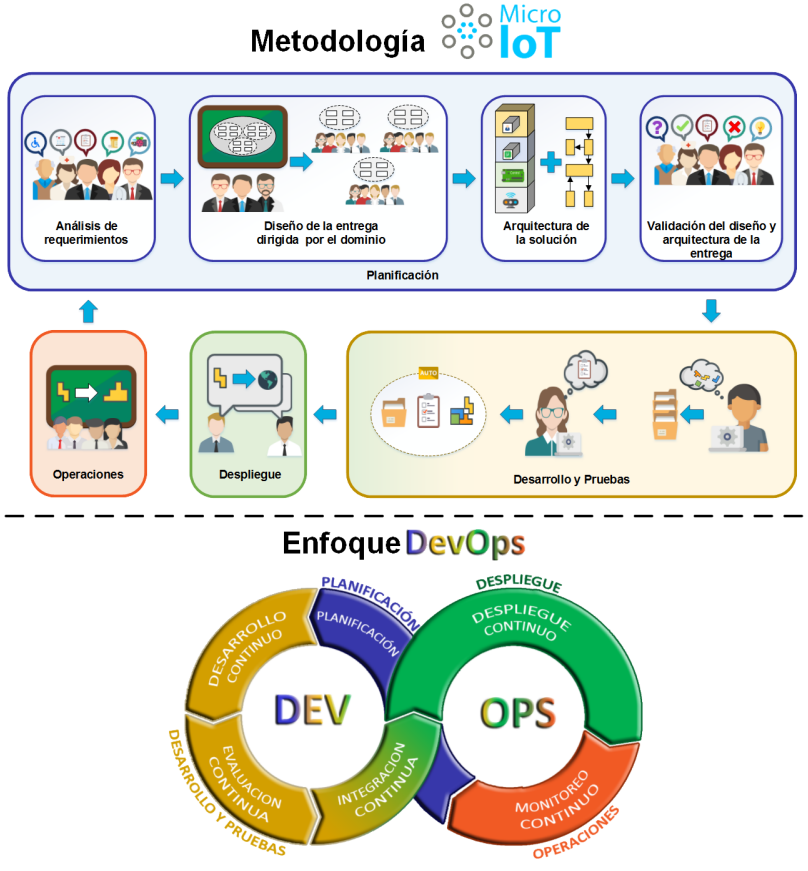


Figura 4.1: Actividades de la metodología MicroIoT y su alineación con DevOps (Fuente: Elaboración propia).

La metodología MicroIoT, abarca siete actividades: i) Análisis de requerimientos, ii) Diseño de la entrega dirigida por el dominio, iii) Arquitectura de la solución, iv) Validación del diseño y arquitectura de la entrega, v) Desarrollo y Pruebas, vi) Despliegue y vii) Operaciones. El ciclo de vida de MicroIoT, se muestra en la Figura 4.2, como ya se explicó, se trata de una metodología ágil, basada en un desarrollo iterativo e incremental, en el cual, en cada iteración se obtiene una nueva entrega que agrega funcionalidad y valor al sistema de software IoT para AAL.

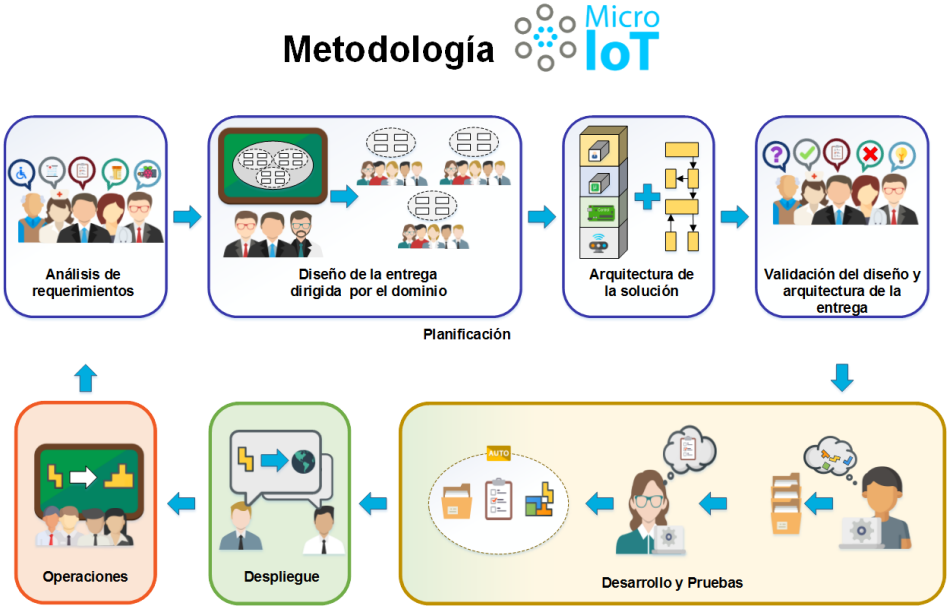


Figura 4.2: Actividades de la metodología MicroIoT y su alineación con DevOps (Fuente: Elaboración propia).

Cada una de las actividades planteadas por MicroIoT, implica una serie de tareas y pasos a seguir. El Anexo A.4 muestra el correspondiente diagrama de procesos SPEM, el mismo indica los artefactos de entrada y salida de cada actividad, así como los roles involucrados en su desarrollo. Para una mejor comprensión de este tipo de diagramas en el Anexo A.3 presenta un resumen de la nomenclatura aplicada. La Tabla 4.1 muestra: en la primera columna, las siete actividades principales de MicroIoT, en la segunda columna, sus respectivas tareas y en la tercera columna, los roles involucardos, que, son quienes realizan dichas tareas.

Actividades de la Metodología	Tareas de la Actividad	Roles Involucrados
1. Análisis de Requerimientos	Elicitación de Requerimientos Priorización de requerimientos Definición de entregas	- Analista de Requerimientos - Interesados - Analista de Requerimientos - Interesados - Analista de Requerimientos - Interesados - Arquitecto de Software
2. Diseño de la Entrega Dirigido por el Dominio	2.1 Análisis de dominio	- Analista de Requerimientos - Arquitecto de Software - Interesados
	2.2 Delimitación de contextos	- Analista de Requerimientos - Equipo Principal - Arquitecto de Software
	2.3 Definición de entidades, agregados y servicios	- Arquitecto de Software - Equipo Principal
	2.4 Identificación, definición y verificación de microservicios	- Arquitecto de Software - Equipo Principal
	2.5 Constitución de equipos de trabajo	- Arquitecto de Software - Equipo Principal
	2.6 Establecimiento de modelos de datos	- Equipos Multifuncionales
3. Arquitectura del sistema o entrega	3.1 Adecuar la arquitectura de la aplicación 3.2 Establecer la arquitectura de gestión de microservicios	- Arquitecto de Software - Equipo Principal - Arquitecto de Software - Equipo Principal
4. Validación del Diseño y Arquitectura de la Entrega	4.1 Validación del diseño y de la arquitectura de la solución.	- Analista de Requerimientos - Arquitecto de Software - Equipo Principal - Interesados
5. Desarrollo y pruebas	5.1 Codificación y construcción de la aplicación 5.2 Evaluación continua 5.3 Integración continua	- Equipo de Desarrollo - Equipo de Qa. - Equipo de Desarrollo. - Equipo de Operaciones
6. Despliegue	6.1 Despliegue continuo	- Equipo de Operaciones

Actividades de la Metodología	Tareas de la Actividad	Roles Involucrados
7. Operaciones	7.1 Monitoreo continuo	- Equipo de Operaciones
	7.2 Retroalimentación de los Interesados	- Arquitecto de Software - Analista de Requerimientos - Interesados - Equipo Principal

Tabla 4.1: Actividades, tareas y roles de la metodología.

A continuación se presenta la definición de cada uno de los roles involucrados, en las actividades que MicroIoT considera necesarias, para llevar a cabo el desarrollo y mantenimiento de un sistema de IoT para AAL.

4.4.1. Definición de roles involucrados

En las diferentes actividades planteadas en MicroIoT, la metodología propuesta, intervienen diferentes actores o roles que deben cumplir las personas que desarrollen dichas actividades. Los principales roles que intervendrán durante el ciclo de vida planteado para cada entrega de software, en MicroIoT, pueden ser de dos tipos, roles internos como el caso de los roles de la organización de desarrollo, o externos como lo son los interesados, conocidos tambien como *Stakeholders*.

A. Roles de la organización de desarrollo

La organización de desarrollo de sistemas de software para Internet de las Cosas debe contar con personal que cumpla los siguientes roles:

I Arquitecto de Software

II Analista de Requerimientos

III Equipo de Desarrollo (Development): El equipo de desarrollo abarca varios roles, entre los más comunes están:

- Desarrollador
- Evaluador (*Tester*)
- Diseñador
- Administrador de base de datos

IV Equipo de Aseguramiento de la Calidad (Qa)

V Equipo de Operaciones (Ops)

B. Interesados

Otro grupo de personas muy importante son los interesados, una persona que cumple el rol de interesado puede ser cualquiera de los siguientes:

I Cliente (puede ser el gerente de la organización cliente, director de un hospital, los usuarios finales, etc.)

II Experto del dominio

- Experto en IoT
- Experto en Salud

En la Tabla 4.2, se encuentra la descripción de cada uno de los roles mencionados anteriormente, junto con las respectivas laboress que cada rol debe desempeñar.

Rol	Descripción	Labores
Interesados	Los interesados son los clientes, éstos pueden ser: un experto de dominio, un gerente de la organización cliente o usuarios finales del sistema de IoT para AAL.	1. Expresar los requerimientos, el problema de negocio, y los recursos de los que dispone.
		2. Priorizar requerimientos, junto con el Analista de Requerimientos y el Arquitecto de Software.
		3. Establecer las entregas.
		4. Validar la solución propuesta.
Analista de Requerimientos	El analista de requerimientos es la persona que realiza un levantamiento de requisitos. Toma decisiones junto con los interesados.	1. Obtener los requerimientos del sistema, evitando ambigüedades.
		2.Establecer las entregas y sus requerimientos.
		3. Definir el dominio y profundizar la elicitación en los puntos claves.
		4. Identificar recursos, prioridades y evitar asunciones de los involucrados.

Rol	Descripción	Labores
Arquitecto de Software	Es el líder del proyecto, responsable del proceso global, de la coordinación y aplicación efectiva de la metodología "MicroIoT".	1. Establecer las entregas.
		2. Diseñar la entrega.
		3. Establecer microservicios.
		4. Establecer la arquitectura del sistema.
		5. Validar la solución propuesta.
		6. Designar un equipo multifuncional a cada microservicio.
		7. Vigilar que se cumplan adecuadamente las tareas y actividades de MicroIoT.
Equipo de Desarrollo (Dev)	Encargados de generar un producto de código a partir de los requerimientos. Pueden desempeñar diferentes roles p. ej. desarrollador, evaluador, diseñador GUI, administrador de BD, entre otros.	1. Cumplir entregas en un plazo establecido.
		2. Cumplir con los requerimientos del sistema
		3. El desarrollador genera el código fuente de un microservicio.
		4. El evaluador ejecuta las pruebas diseñadas por el Equipo de Qa.
		5. El administrador de base de datos, modela, gestiona y mantiene la base de datos.
		6. El diseñador analiza y mejora la interacción del sistema con el usuario.
Equipo de Operaciones (Ops)	Encargados de poner el software en producción; además, supervisa el entorno de despliegue.	1. Colocar en producción las entregas o funcionalidades.
		2. Automatizar tareas de despliegue y monitoreo mediante herramientas.
		3. Configurar el entorno de producción.
		4. Mantener la estabilidad y rendimiento de la infraestructura del sistema.

Rol	Descripción	Labores
Equipo de Aseguramiento de la Calidad (Qa - Quality Assurance)	Tienen como objetivo garantizar que el producto de software tenga alta calidad y resiliencia. Esta tarea se facilita con la automatización, de la evaluación continua	1. Asesorar al equipo de desarrollo para asegurar la calidad mediante pruebas unitarias.
		2. Cumplir con los requerimientos de calidad.
		3. Plantear pruebas de funcionalidad.
		4. Reportar las deficiencias encontradas.
Equipo Multifuncional	Son los equipos de trabajo, se encargan del desarrollo, despliegue, validación, verificación y mantenimiento de uno o más microservicios, son multidisciplinarios o de funcionalidad cruzada, dado que están conformados por miembros del Equipo Dev, Equipo Ops, y miembros del Equipo de Qa.	1. Desarrollar y desplegar la aplicación al entorno de producción.
		2. Establecer los servicios de un microservicio en base a lo acordado con el equipo principal.
		3. Cumplir los requerimientos establecidos para el microservicio a construir.
		4. Validar y verificar los microservicios de los cuales están a cargo.
Representante de Equipo	A pesar de que en un equipo multifuncional no existe una jerarquía establecida, es necesario contar con un representante de equipo, este será designado por el Arquitecto de Software.	1. Comunicar al equipo principal y al Arquitecto de software las necesidades, avances e inconvenientes presentados en el equipo funcional y viceversa.
Equipo Principal	Está constituido por los representantes de cada Equipo Multifuncional. Su objetivo es validar y verificar el proceso de desarrollo y comunicar necesidades o avances al Arquitecto de software.	1. Establecer estándares para brindar transparencia a los usuarios finales.
		2. Ayudar en las tareas del Arquitecto de Software.
		3. Reunirse con el Arquitecto de Software para la definición y asignación de nuevas entregas.

Tabla 4.2: Roles involucrados en MicroIoT y descripción de sus labores.

Dada la descripción de los roles, se presenta un resumen en la Figura 4.3, la cual ilustra el diagrama de roles involucrados en las actividades y fases de la metodología MicroIoT. Se presentan los roles a modo de herencia, dado que un rol que esté en un nivel superior abarca a todos aquellos que hacen referencia a él mediante una flecha.

Nota: una persona puede cumplir uno o varios de estos roles, siempre y cuando cuente con los conocimientos necesarios y suficientes para cubrir el rol o roles asignados. La importancia de realizar adecuadamente las tareas asignadas a cada rol es muy importante para obtener un sistema exitoso.

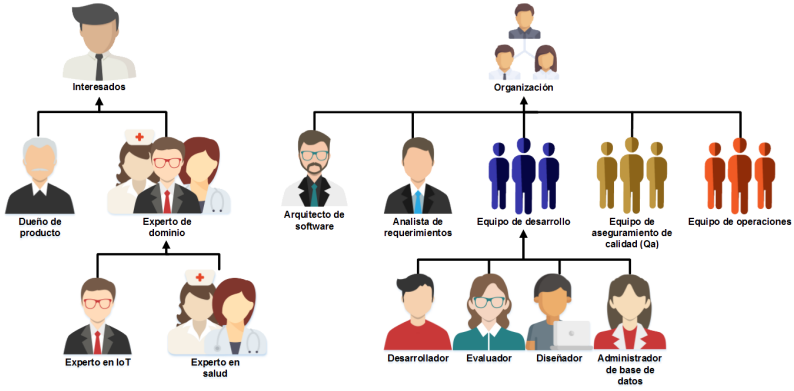


Figura 4.3: Diagrama de roles que intervienen en las actividades planteadas en MicroIoT (Fuente: Elaboración propia).

Antes de desarrollar o codificar un sistema de software, se necesita una visión general del sistema que se va a crear, identificar los requerimientos y diseñar la solución sin preocuparse por detalles de implementación ni por las tecnologías que se utilizarán (Wasson, 2017), por ello; se considera la actividad de Análisis de Requerimientos y posteriormente la actividad de Diseño de la entrega Dirigido por el Dominio (DDD).

4.4.2. Análisis de Requerimientos

El análisis de requerimientos es vital para la planificación del desarrollo de una solución o sistema de software, dado que la metodología propuesta se centra en un desarrollo ágil se consideran iteraciones cíclicas para obtener entregas continuas a corto tiempo y reducir el tiempo de puesta en producción de un producto en el mercado o una entrega que abarca requerimientos priorizados.

Esta actividad se centra en describir la elicitación de requerimientos en un entorno general, y posteriormente la priorización de requerimientos. En caso de ser un sistema pequeño y poco complejo, el mismo podrá ser desarrollado en una sola entrega. La Figura 4.4 ilustra el diagrama de procesos de la actividad "Análisis de Requerimientos" que abarca tres tareas: i) Elicitación de requerimientos, ii) Priorización de requerimientos, y iii) Definición de entregas, las cuales se describen a continuación.

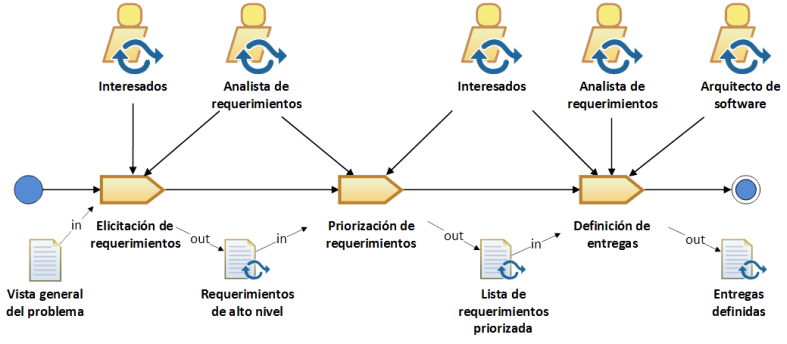


Figura 4.4: Tareas de actividad: "Análisis de Requerimientos" (Fuente: Elaboración propia).

A. Elicitación de requerimientos

La elicitación de requerimientos tiene como objetivo obtener una vista general del problema o solución de IoT para AAL identificando: i) las funcionalidades generales tanto de las cosas, como de software que las controlará y obtendrá la información, y ii) entregables requeridos, a modo de requerimientos de alto nivel. Esto servirá para que más adelante, por medio de la vista general obtenida, el Arquitecto de Software pueda identificar el tamaño del sistema y definir el número de entregas necesarias para obtener el sistema completo.

El Analista de Requerimientos debe profundizar la elicitación hasta llegar a un nivel de detalle suficiente como para determinar el tamaño del sistema, en este proceso no es necesario profundizar en requerimientos no funcionales o requerimientos de hardware, dado que la elicitación de estos aspectos se profundizará más adelante en el análisis del dominio, es más importante centrarse en las funcionalidades generales y entregables que los interesados manifiesten, para así cumplir con el objetivo de esta tarea.

Dado que la creación de aplicaciones de microservicios involucra una arquitectura ágil, MicroIoT recomienda aplicar en esta tarea, las prácticas de

Ingeniería de requerimientos para metodologías ágiles indicadas por Sillitti and Succi (2005), que se describen a continuación:

- Durante el proceso de elicitación interviene el analista de requerimientos y los interesados (experto de dominio, dueño del producto, etc.).
- Durante la elicitación se realiza una lluvia de ideas entre el analista de requerimientos y los interesados, para determinar el problema a resolver, durante esta fase se puede trabajar con bosquejos que faciliten la adquisición de requerimientos.
- Los requerimientos son recolectados utilizando lenguaje natural.
- Se recomienda llevar un documento que resuma los requerimientos identificados durante el proceso de elicitación para revisiones futuras.
- Los requerimientos complejos se deben separar en requerimientos más simples.

Una vez que se han identificado los requerimientos de alto nivel como lo son la vista general del problema o solución de software IoT para AAL, el Analista de Requerimientos junto con los Interesados deben priorizar los requerimientos para definir el orden en el que serán entregados. A continuación se presenta la descripción de tarea de priorización de requerimientos.

B. Priorización de requerimientos

De acuerdo con Wohlin et al. (2005), la priorización es un paso crucial para tomar buenas decisiones con respecto a la planificación de productos para entregas únicas y múltiples. Se consideran diversos aspectos de la funcionalidad, como la importancia, el riesgo, el costo, etc. Las decisiones de priorización las toman los interesados, incluidos los usuarios, los gerentes, los desarrolladores o sus representantes como lo es el analista de requerimientos. Según Wohlin et al. (2005), algunos de los aspectos relevantes para tomar en cuenta son los siguientes:

- **Importancia:** Al priorizar la importancia, las partes interesadas deben priorizar cuáles son los requisitos más importantes para el sistema. Depende en gran medida de la perspectiva que tenga el interesado. La importancia podría ser, por ejemplo, la urgencia de la implementación, la importancia de un requisito para la arquitectura del producto, la importancia estratégica para la empresa, etc.

II **Penalización:** Es posible evaluar la penalización que se introduce si un requisito no se cumple. Por ejemplo multas a requerimientos que son de baja importancia o que el usuario o interesado da por sentado y que no han sido cumplidas satisfactoriamente.

III **Recursos:** Por lo general el costo es estimado por la organización, considerando medidas como la complejidad del requisito, la capacidad de reutilizar el código existente, la cantidad de pruebas y la documentación necesaria. El costo a menudo se expresa en términos de horas de personal (esfuerzo) El costo y el tiempo podrían priorizarse simplemente estimando el costo real en una escala absoluta o normalizada.

IV **Riesgo:** La gestión de riesgos se utiliza para hacer frente a riesgos internos (técnicos y de mercado) y externos (reglamentaciones, proveedores, etc). La gestión del riesgo también se puede utilizar al planificar los requisitos en las entregas y las liberaciones identificando los riesgos que pueden causar dificultades durante el desarrollo, por ejemplo, riesgos de desempeño, riesgos de procesos, riesgos de cronogramas, etc. Con base en la probabilidad de riesgo estimada y el impacto del riesgo para cada requerimiento, es posible calcular el nivel de riesgo de un proyecto.

V **Volatilidad:** Las razones para la volatilidad de los requisitos varían, por ejemplo: el mercado cambia, los requisitos del negocio cambian, los cambios legislativos ocurren, los usuarios cambian o los requisitos se vuelven más claros durante el ciclo de vida del software. Independientemente de la razón, los requisitos volátiles afectan la estabilidad y la planificación de un proyecto, y presumiblemente aumentan los costos.

Una vez que se han definido las prioridades de los requerimientos de alto nivel, se debe definir las entregas que se harán para el desarrollo del software de IoT para AAL, puede darse en una única entrega o en múltiples entregas, por ello, MicroIoT presenta a continuación la tarea "Definición de entregas", la cual presenta las pautas necesarias.

C. Definición de entregas

En caso de que la solución o sistema de software abarque demasiados contextos o ambientes de despliegue, de modo que no sea posible identificar adecuadamente todos los requerimientos, ya sea, funcionales, requerimientos de hardware, y no funcionales, y además los equipos de trabajo, a criterio del Arquitecto de Software o del Equipo Principal, no puedan cumplir con todos los requerimientos en una sola entrega que debe ser terminada en un plazo de 2 a 4 semanas, lo cual es característico de una metodología ágil.

Las entregas de software son definidas a partir de las prioridades establecidas en la tarea anterior. La primera entrega debe ser un poco más grande que las posteriores, pues esta debe servir de base para todas las demás entregas. Las entregas posteriores pueden ser sistemas independientes o simplemente incremento de funcionalidades en el sistema base de la primera entrega.

La entrega que de aquí en adelante puede ser denominada "sistema de software", sistema o aplicación, dado que esta entrega puede hacer referencia al sistema de software completo o a un subsistema del mismo en caso de que la complejidad y el contexto del mismo sea demasiado amplio como para trabajar sobre el mismo en una sola iteración de esta metodología.

4.4.3. Diseño de la entrega dirigida por el dominio

Un buen diseño permite una óptima aplicación de la Arquitectura de Microservicios para el soporte a nivel de resistencia, escalabilidad, y rendimiento; por ello coincidimos en que es la parte más importante en el ciclo de vida de una aplicaciòn de Microservicios. Aplicar el estilo de diseño de software dirigido por el dominio, luego del proceso de adquisición de requerimientos, permite modelar un sistema acorde a los requerimientos con una perspectiva de escalabilidad y rendimiento.

MicroIoT basa el diseño de la entrega en un proceso recomendado por Wasson (2017), orientado a soluciones de IoT para AAL. Como se vio en la sección 2.6.1, el diseño basado en dominios tiene dos fases distintas: una fase estratégica y una fase táctica.

- Durante la fase estratégica del diseño basado en dominios (DDD - *Domain Driven Design*), se asigna el dominio empresarial y se definen los contextos delimitados para los modelos de dominio.
- La fase táctica consiste en definir los modelos de dominio con precisión, identificando entidades, objetos de valor y agregados, dentro de un cada contexto delimitado.

En la Figura 4.5 se muestran las tareas propuestas para la actividad de Diseño de la Entrega Dirigido por el Dominio, las mismas están alineados a las las recomendaciones de Wasson (2017) y Sharma (2017).

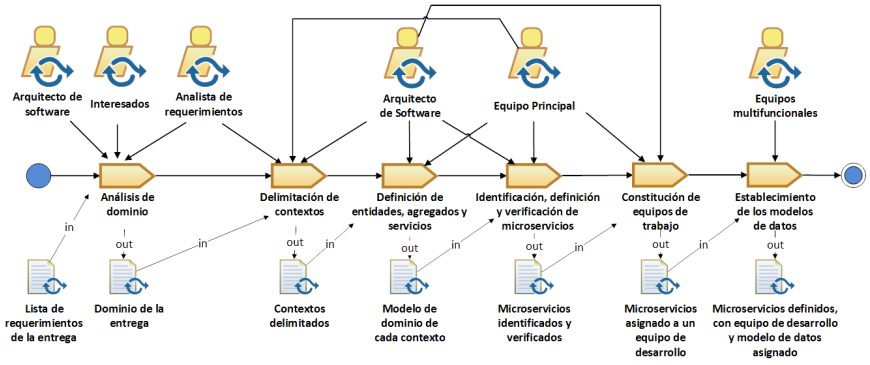


Figura 4.5: Tareas de la actividad "Proceso del Diseño de la entrega dirigida por el dominio" (Fuente: Elaboración propia).

Como se ilustra en la Figura 4.5 que muestra el diagrama de procesos, la actividad de "Diseño de la Entrega Dirigida por el Dominio" comprende las tareas de: i) Análisis de Dominio, ii) Delimitación de Contextos, iii) Definición de entidades, agregados y servicios, iv) Identificación y Verificación de microservicios, v) Constitución de equipos de trabajo, y vi) Establecimiento de los modelos de datos. Como se describe a continuación.

A. Análisis del Dominio

El Analista de Requerimientos junto con el Arquitecto de Software del proyecto deben analizar el dominio en el que se desarrollan los requerimientos, identificando: controladores de hardware, el contexto del sistema, y los factores que los interesados estimen críticos para el éxito.

Para soluciones de software de AAL *(Ambient Assisted Living)*, además de identificar requerimientos funcionales, se recomienda enfatizar en identificar las características generales de AAL mostrados en la Tabla 4.3, y los aspectos de calidad de software, mostrados en la Tabla 4.4, estas características y aspectos se basan en los resultados obtenidos en la revisión sistemática realizada en el Capítulo 3.

Las caracteristicas de IoT y AAL tanto de la Tabla 4.3 como los aspectos de calidad de la Tabla 4.4, deben ser considerados dentro del análisis del dominio, pero esto no implica que deban ser los únicos, puede ser que el dominio implique profundizar en otros requerimientos. El analista de requerimientos y el arquitecto de software deben definir y analizar adecuadamente el dominio.

A continuación, se definen los contextos y se presentan recomendaciones para identificarlos con facilidad.

Requerimientos Generales de AAL Identificados	
Tipo o tipos de servicio de salud que abarca el sistema.	**Orientación del sistema (usuarios finales)**
- Medicación. - Asistencia para adultos mayores. - Asistencia de estilo de vida. - Manejo de enfermedades crónicas. - Vigilancia - Otros	- Paciente - Cuidador o Familiar - Profesionales de la salud - Otros
Requerimientos de Hardware (Dispositivos de Internet de las Cosas disponibles o necesarios)	**Ambiente en el que opera el sistema**
- Dispositivos controladores - Dispositivos sensores - Dispositivos actuadores - Dispositivos inteligentes con sensores, actuadores y controladores integrados - Otros	- Hogar - Hospital - Ubiquitous Health - Otro
Plataforma de despliegue de la o las aplicaciones del sistema	
(para cada plataforma de despliegue de la o las aplicaciones es importante considerar a quién está orientada o quién será el usuario final de esa aplicación).	
- Aplicación Web - Aplicación Móvil - Aplicación de Escritorio - Otro	

Tabla 4.3: Características Generales de AAL

Aspectos de Calidad de Software Identificados	
Aspectos de calidad según ISO 25010	**Estándares de Salud**
- Seguridad. - Usabilidad. - Interoperabilidad. - Eficiencia. - Disponibilidad. - Mantenibilidad. - Autenticidad. - Otro.	- HL7 Reference Information Model - HL7 Electronic Health Record - ISO 13940 (ContSys) - Otro

Tabla 4.4: Requerimientos de calidad generales en AAL.

B. Delimitación del contexto

La delimitación del contexto está a cargo del Analista de Requerimientos y el Arquitecto de Software. Un contexto delimitado establece el límite o un ámbito a abarcar dentro de un dominio, en donde se aplica un modelo de dominio en particular. Una vez delimitado el contexto se establece su modelo de dominio que representa un subdominio concreto de la aplicación mayor o una vista arquitectónica que representa el ámbito de la solución a ser creada (Wasson, 2017).

Nota: En caso de existir contextos previamente identificados (de una entrega o iteración anterior), el Arquitecto de Software deberá hacer un mapeo para verificar si los nuevos cambios afectan o no a los contextos pre-existentes y adaptar los cambios en el subdominio respectivo, tomando decisiones junto con el equipo principal.

Para identificar contextos en el dominio de Internet de las Cosas para Ambientes de Vida Asistidos se recomienda considerar los requerimientos levantados en el Análisis de Dominio Tabla 4.4, entre los más importantes a considerar para determinar posibles contextos están:

- *Servicio de salud que abarca el sistema.*
- *Ambiente en el que opera el sistema.*

Además, es necesario identificar las entidades con sus atributos y funciones, para establecer el modelo de diseño de la entrega guiado por el dominio. A continuación se presentan criterios para establecer estos componentes de DDD.

C. Definición de entidades, agregados y servicios.

Como se estableció en la Sección 2.6.2, dentro de un contexto delimitado las entidades deben ser definidas (objetos con una identidad única que persisten en el tiempo), objetos de valor (no tiene identidad, se define únicamente mediante los valores de sus atributos), agregados (entidad raíz, cuyo tiempo de vida define el tiempo de vida de sus hojas o heredados) y servicios de dominio (funcionales o no funcionales proporcionados por una entidad), esta tarea está a cargo del Arquitecto de Software. A continuación, se presentan algunas sugerencias para identificarlos.

I Descomponer la aplicación por sustantivo, esto ayuda a identificar las entidades y los objetos de valor.

   a. Se debe considera que no todos los sustantivos son entidades u objetos de valor.

   b. Para el dominio de IoT para AAL, por lo general, las entidades pueden ser identificadas dentro de los siguientes características generales de la Tabla 4.3:

      - *Orientación del sistema (usuarios finales).*
      - *Ambiente en el que opera el sistema.*
      - *Dispositivos de IoT (disponibles o necesarios).*

   c. Para el dominio de IoT para AAL, por lo general, los objetos de valor resultan ser atributos de las entidades, si un atributo es complejo se puede identificar con un objeto de clase, o "entidad" sin identificador. Por ejemplo, la información de contacto, puede ser un objeto de valor de una entidad **Paciente**.

II Usar la descomposición basada en verbos y definir servicios que implementan un caso de uso único. Un ejemplo típico es un flujo de trabajo que implica varios microservicios.

   a. No todos los verbos son servicios.

   b. Los servicios generalmente resultan ser las funcionalidades de las entidades y objetos de valor.

III Identificar Agregados: La identificación de agregados (artefactos de DDD), es muy importante para identificar a los microservicios más adelante, por ello, a continuación, se presentan algunas pautas para identificar agregados en un modelo de dominio.

a. Los agregados hacen referencia a aquellas entidades raíz, de las cuales otras entidades se derivan (entidades padre o entidades que tienen componentes).
b. Un agregado también se identifica por el tiempo de vida de la entidad si una entidad raíz desaparece, sus componentes también (Sharma, 2017).
c. Un agregado es un límite de persistencia (Wasson, 2017).
d. Los agregados deben tener un acoplamiento flexible (Wasson, 2017).

Es recomendable bosquejar el modelo de dominio. El modelo de dominio incluirá representaciones de cosas del mundo real, como son usuarios, dispositivos de hardware, dispositivos embebidos, sensores, entre otras. Pero no todas las partes del sistema deben usar las mismas representaciones para las mismas cosas, por ejemplo: dos sensores pueden trabajar de forma independiente. Si se crea un modelo único para ambos subsistemas, sería innecesariamente complejo haciendo más difícil que el modelo evolucione con el tiempo, porque los cambios deberán satisfacer a varios equipos que trabajen en subsistemas independientes. Por lo tanto, a menudo es mejor diseñar modelos independientes que representen la misma entidad del mundo real, en dos contextos diferentes (Vernon, 2013).

D. Identificación, Definición y Verificación de Microservicios

Según Vresk and Čavrak (2016) cada microservicio es responsable de una o más funciones estrechamente relacionadas.

I **Identificación de microservicios**

Los microservicios serán definidos a partir de los resultados obtenidos en las tareas anteriores de esta actividad. Como principio general, un microservicio no debe ser menor que un agregado ni mayor que un contexto delimitado (Wasson, 2017); además, se debe tener en cuenta que los agregados suelen ser buenos candidatos para microservicios. Los requisitos no funcionales, como el tamaño del equipo, los tipos de datos, tecnologías, escalabilidad, disponibilidad y seguridad, pueden volver a dividir un microservicio en dos o más microservicios más pequeños, o a hacer lo contrario combinando varios microservicios en uno.

II **Definición de microservicios**

Un detalle importante a tener en cuenta en un microservicio según Uviase and Kotonya (2018), es que un microservicio clasifica a los servicios que ofrece en:

- Servicios funcionales: servicios que soportan las funciones operativas de un sistema inteligente en un dominio de IoT. En IoT, estos servicios son en su mayoría literales (es decir, números, letras, etc.). Estos servicios están expuestos para ser utilizados por un sistema o dispositivo externo.
- Servicios no funcionales: servicios que están relacionados con tareas no operativas (p. ej. registro, autenticación, monitoreo, creación, auditoría) pero deben ser utilizados para una operación confiable del sistema. Estos servicios no están expuestos y son utilizados por un sistema o dispositivo que desea integrarse en un sistema de IoT existente.

Con base en esa clasificación, se definen a continuación los componentes que regularmente abarca un microservicio cuando se pretende implementar sistemas de IoT para AAL basados en microservicios.

a. **Front-end:** Este componente hace referencia las interfaces gráficas de usuario, éstas pueden ser web, móviles o de escritorio.

b. **Base de datos:** Cada microservicio abarca un subdominio del sistema de IoT para AAL al que pertenece; por ello, debe manejar sus datos independientemente (aunque comparta la base de datos, usa tablas exclusivas para ese microservicio), en las tareas posteriores se definirá un modelo de datos para cada microservicio.

c. **Back-end:** Este componente hace referencia a la lógica de negocio, su implementación a nivel de código está basado el submodelo de dominio correspondiente, se encarga de la gestión base de datos, cálculos, entre otros. Por lo general, cuando se codifica, está dispuesto en capas: capa de dominio (Dom), capa de acceso a datos (Dao) y capa de servicios (Serv).

d. **Sistemas externos de IoT:** Estos son sistemas que consumen o proveen servicios funcionales al microservicio. Éstos están compuestos por controladores y elementos de hardware que pueden ser sensores o actuadores; también, se consideran dispositivos que tienen controladores y sensores integrados, por citar un ejemplo, puede ser un teléfono celular inteligente. En un entorno de internet de las cosas, estos sistemas poseen una lógica de negocio o back-end e incluso pueden abarcar un front-end, en caso de que este sistema tenga que interactuar con el usuario.

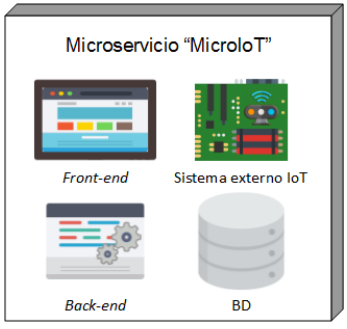


Figura 4.6: Definición de un microservicio para un entorno IoT (Fuente: Elaboración propia).

Todos los componentes deben ser considerados por un microservicio que pertenece a un sistema de IoT para AAL, en la Figura 4.6, se ilustra gráficamente esta definición de componentes de un microservicio.

III **Verificación de microservicios**

Una vez que el software se ha diseñado y se identificaron los microservicios, es importante que el arquitecto de software revise el modelo para verificar la correcta identificación de los microservicios, Los siguientes criterios están basados en las recomendaciones de Wasson (2017) y Uviase and Kotonya (2018).

a. Cada microservicio tiene una única responsabilidad y se conforma de frontend, backend y su lógica de negocio.

b. No hay llamadas que generen una alta conexión y dependencia entre microservicios. Si dividir la funcionalidad en dos microservicios hace que estos generen demasiada conversación, esto puede ser una indicación de que estas funciones deben estar en el mismo microservicio.

c. Cada microservicio es lo suficientemente pequeño como para que un equipo pequeño lo pueda generar trabajando de forma independiente.

d. No hay interdependencias que requieran la implementación de dos o más microservicios en sincronía. Siempre debe ser posible implementar un microservicio sin tener que volver a implementar ninguno de los demás microservicios.

e. Los microservicios no están estrechamente acoplados y pueden evolucionar independientemente.

f. Los límites del microservicio no causarán problemas con la coherencia de datos o la integridad.

g. Todas las entidades aparecen exactamente una vez en el esquema. Otras entidades pueden contener referencias a él, pero no lo duplican.

E. Constitución de equipos de trabajo multifuncionales

De acuerdo con Lewis and Fowler (2014) un error frecuente cuando se busca dividir una gran aplicación en partes, es que, a menudo la administración se enfoca en la capa de tecnología, lo que lleva a formar equipos de desarrollo de interfaz de usuario, equipos lógicos del lado del servidor y equipos de bases de datos; de ese modo, incluso los cambios simples requieren tiempo y aprobación presupuestaria. Por este motivo, es fundamental una buena división de los equipos de trabajo que se encargaran de determinados microservicios, aunque es importante resaltar que un mismo equipo de trabajo puede trabajar sobre uno o más microservicios, y además es primordial, que cada uno de los microservicios tenga un equipo de trabajo a cargo del desarrollo y mantenimiento del mismo.

La Conway (1968) declara: *"Cualquier organización que diseña un sistema producirá un diseño cuya estructura es una copia de la estructura de comunicación de la organización"*. Por lo tanto, los componentes del dominio y los equipos son sinónimos. Este enfoque corresponde a la idea de equipos multifuncionales (*cross-functional*), la coordinación entre equipos no se requiere con mucha frecuencia, ya que las funciones se implementan idealmente por equipos independientes. Si el objetivo es desarrollar componentes de dominio individuales e independientes el uno del otro, es decir Microservicios, entonces, en función de los microservicios, se pueden crear equipos de trabajo multifuncionales (Wolff, 2016). Un ejemplo se muestra en la Figura 4.7.

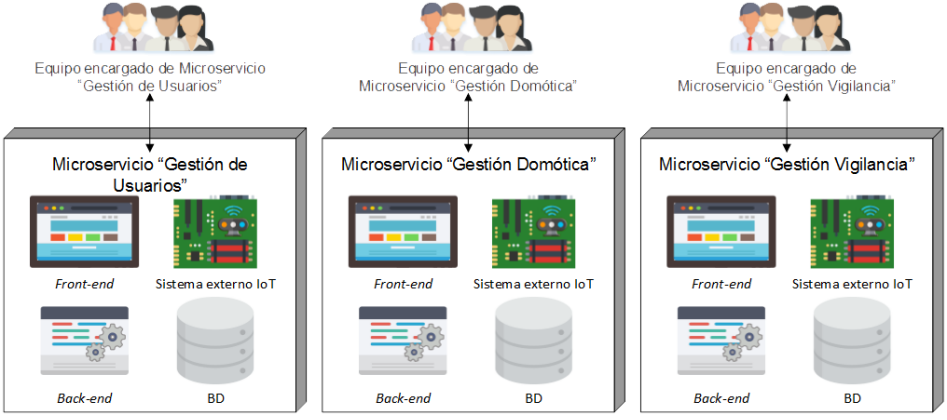


Figura 4.7: Proyecto por Dominios con MicroIoT (Fuente: Elaboración propia).

La Figura 4.7 ilustra equipos individuales para la gestión de usuarios (médicos, pacientes, enfermeras), gestión de controladores del hogar y otro equipo para vigilancia personalizada. Estos equipos multifuncionales o de funcionalidad cruzada, trabajan en sus respectivos microservicios, abarcando *front-end, back-end* y gestión de base de datos.

I **Formación de equipos multifuncionales y equipo principal**

El Arquitecto de Software se encarga de crear los equipos de trabajo, como se describe en la Tabla 4.2, los equipos de trabajo deben ser equipos multifuncionales, con miembros del grupo de Desarrollo (*Development*), miembros del grupo de Aseguramiento de la Calidad (*Qa - Quality assurance*), y miembros del equipo de Operaciones (*Operations*).

El Arquitecto de Software debe establecer un Equipo Principal (*Core Team*) que está conformado por un representante de cada equipo multifuncional, el Arquitecto de Software selecciona el representante de cada equipo de trabajo multifuncional, según el área a la cual pertenezca el problema o motivo que suscita la reunión del Arquitecto de Software con los representantes de cada equipo, es decir, sea un asunto de desarrollo, operaciones, o calidad.

En la Figura 4.8 se observa la formación de los equipos funcionales a partir de los equipos *Development, Operations y Quality assurance* y la formación del Equipo Principal (*Core Team*) a partir de los Equipos Multifuncionales (*Cross-functional Teams*).

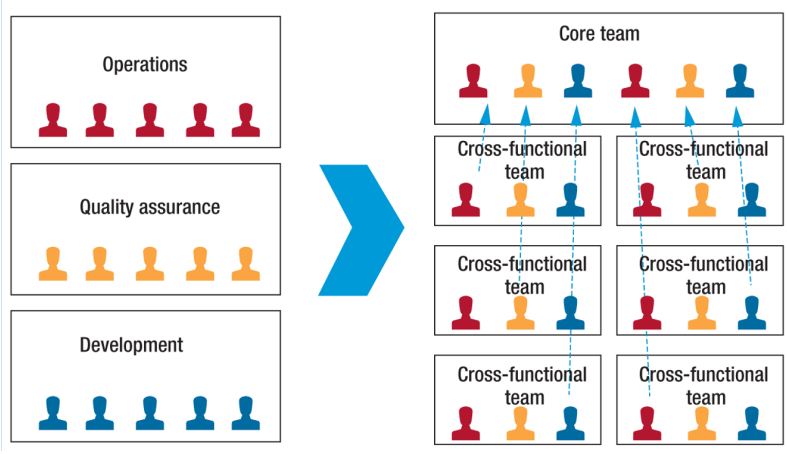


Figura 4.8: Formación de equipos multifuncionales y equipo principal (Fuente:(Balalaie et al., 2016)).

Los equipos multifuncionales deben estar conformados con un hasta un máximo de diez personas, como se menciona en Kim et al. (2016), quien sugiere aplicar la "regla de las dos pizzas" que establece que el número ideal de integrantes que conforman un equipo descentralizado, es aquel que: en la hora de receso puede alimentarse de dos pizzas, el límite del tamaño del equipo influye en tres aspectos:

1. Descentraliza el poder, facilita la autonomía.
2. Se ayuda a garantizar un entendimiento claro del sistema en el que el equipo trabaja.
3. Cada integrante obtiene experiencia de liderazgo en un entorno en donde las fallas no tienen consecuencias catastróficas

Durante el desarrollo del ciclo de vida de MicroIoT, diversos roles están involucrados además de los equipos de trabajo que se encargan de los microservicios, entre ellos están: el Analista de Requerimientos, el Arquitecto de Software y los Interesados (*StakeHolders*) junto con el equipo principal, como se presenta en la Figura 4.9.

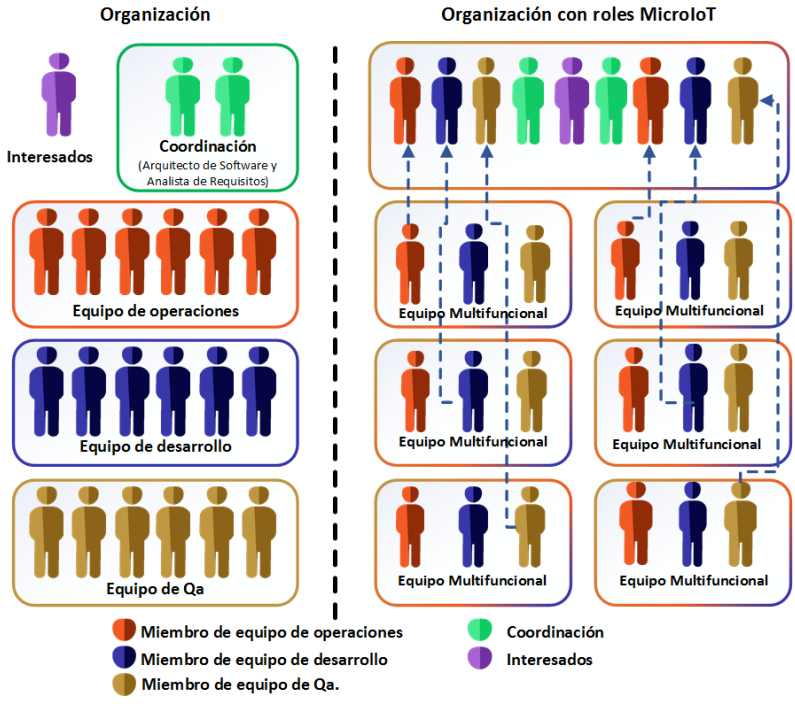


Figura 4.9: Formación de equipos multifuncionales y equipo principal con roles MicroIoT (Fuente: Elaboración propia).

II **Asignación de equipos de trabajo**

Como ya se mencionó un equipo de trabajo debe encargarse de uno o más microservicios, el encargado de asignar el o los microservicios a cada equipo de trabajo es el Arquitecto de Software, se recomienda que los microservicios asignados a un mismo equipo de trabajo pertenezcan a un mismo contexto o contextos parecidos para que el equipo se sienta más familiarizado con el lenguaje ubico de cada microservicio, también se podría considerar unificar a los microservicios de un mismo equipo de trabajo multifuncional, en caso de ser posible.

F. Establecimiento de los modelos de datos

Cada microservicio administra sus propios datos, la integridad y la coherencia de los datos son desafíos fundamentales. Un principio importante que

debe tomarse en cuenta en la administración de datos, como lo menciona Wasson (2017), es que cada servicio administra sus propios datos. Dos servicios no deben compartir un mismo almacén de datos. En su lugar, cada servicio es responsable de su propio almacén de datos privado, al que otros servicios no pueden acceder directamente" Éste principio se ilustra en la Figura 4.10.

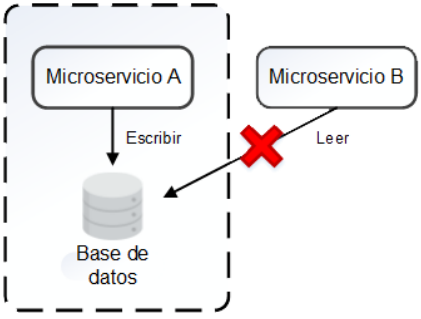


Figura 4.10: Consideraciones de bases de datos compartidas (Fuente: Elaboración propia).

1. **Modelo de datos en Internet de las Cosas**

   Los sistemas de IoT operan sobre datos recolectados por distintos dispositivos de hardware, en su mayoría, sensores de diferentes tipos, encargados de receptar información y transmitirla a la nube para que el microservicio encargado opere sobre esta información. De acuerdo con Vresk and Čavrak (2016), los datos están en el corazón de cada sistema de información. Sin dispositivos conectados capaces de capturar, transferir, analizar, informar y actuar sobre los datos, los beneficios del IoT no serían alcanzables y para establecer un modelo de dominio define los siguientes conceptos para un mejor entendimiento del mismo.

   - **Una cosa,** puede ser cualquier objeto, incluso un contexto de datos abstracto en el mundo real. Las cosas se representan como *dispositivos lógicos que tienen una identificación única, nombre, tipo, atributos y una lista de nodos lógicos.*
   - **Un dispositivo lógico,** contiene uno o más nodos lógicos, que representan varios componentes del dispositivo físico.
   - **Un recurso,** es un elemento computacional que proporciona acceso a un elemento de dispositivo lógico particular. Para representar un dispositivo físico como un dispositivo lógico, el desacoplamiento

en los componentes virtuales correspondientes en la mayoría de los casos no proporcionará suficientes datos relevantes.

- **Los atributos,** son parte del modelo de información utilizado para la descripción del recurso, y pueden incluir datos contextuales adicionales, como la ubicación del recurso, la precisión del sensor, etc.

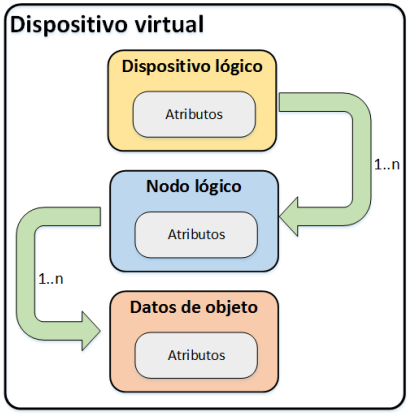


Figura 4.11: Jerarquía del modelo de datos del dispositivo lógico (Fuente: (Vresk and Čavrak, 2016))

En la Figura 4.11 se presenta un modelo de datos jerárquico para IoT, este modelo de datos jerárquico planteado por Vresk and Čavrak (2016), se ejemplifica a continuación para proveer un mayor entendimiento.

El dispositivo de IoT modelado, podría contener un dispositivo lógico de medidor inteligente de energía térmica, extendido con la temperatura ambiental circundante y una estimación de uso de energía térmica por día. El dispositivo lógico consistiría en dos nodos lógicos: *medición y estimación.*

- *El nodo lógico de medición* contendría las propiedades del entorno circundante, como:
  - El objeto de datos de temperatura del entorno, y
  - El conjunto de datos del medidor inteligente de energía térmica.
- *El nodo lógico de estimación* contendría un objeto de datos de estimación de uso de energía térmica por delante que representa una

salida de datos de análisis realizada por un microservicio concreto a través de entradas de datos recopiladas de otras salidas de datos de otros microservicios.

El mayor desafío que surge de la digitalización de los activos físicos es preservar el suficiente nivel de detalle. Los *datos* se pueden definir como una representación de hechos, conceptos o instrucciones de una manera formal adecuada para la comunicación, interpretación o procesamiento por máquina humana o electrónica. Cuando los datos se procesan, interpretan, organizan, estructuran o presentan de manera que sean significativos o útiles, se denominan información. Dicha información proporciona un contexto para los datos y, por lo tanto, se puede representar con un *conjunto de tipos de datos primitivos abstractos* que se utilizan para definir objetos de datos en un modelo conceptual virtual.

4.4.4. Arquitectura del sistema o entrega

En esta actividad, la solución planteada se adapta a una arquitectura general de IoT basada en microservicios, es importante recalcar que la arquitectura, puede ser el resultado de combinar una arquitectura de microservicios con otra arquitectura propia del dominio de la solución o de una determinada entrega, para ello; se deben tomar en cuenta los resultados obtenidos en la tarea de "Diseño de la entrega" (sección 4.4.3). Al finalizar esta actividad, se obtiene la infraestructura y el establecimiento de la arquitectura del sistema de software (se recomienda seguir esta misma arquitectura en las siguientes entregas o iteraciones de la metodología), esto incluye la aplicación de patrones propios de la arquitectura de microservicios. Las tareas de esta actividad son dos: i) Adecuación de la arquitectura de la aplicación y ii) Establecimiento de la arquitectura de gestión de microservicios. El diagrama de procesos, se ilustra en la Figura 4.12.

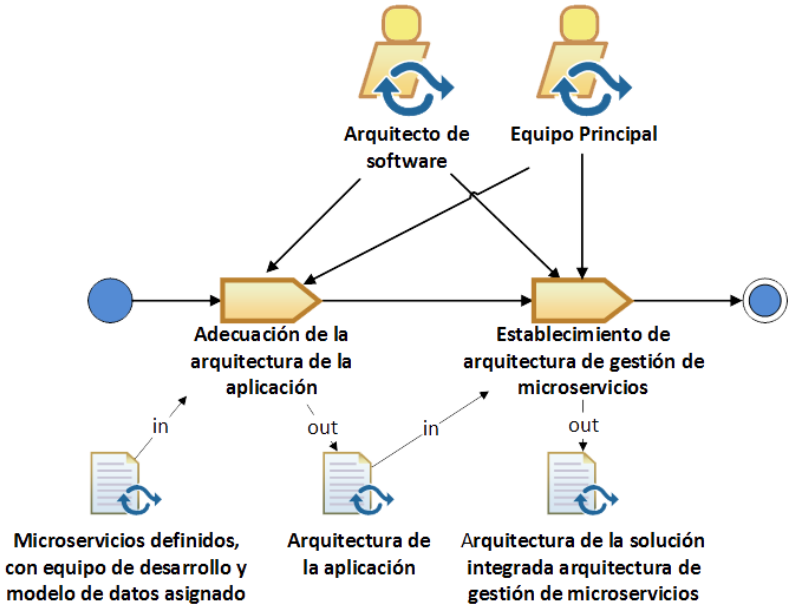


Figura 4.12: Fases de Actividad: "Arquitectura de la solución" (Fuente: Elaboración propia)

A. Adecuación de la arquitectura de la aplicación

Durante esta tarea, se establece la arquitectura de la aplicación a desarrollar en esta entrega, para ello; se propone una arquitectura que permita la interacción de microservicios con entornos IoT, basándose en la arquitectura tradicional de aplicaciones de microservicios combinada con una arquitectura IoT general; ambas, revisadas en el Capítulo 2. En la Figura 4.13Figura 4.13 se muestra una comparación entre la arquitectura general y la arquitectura planteada.

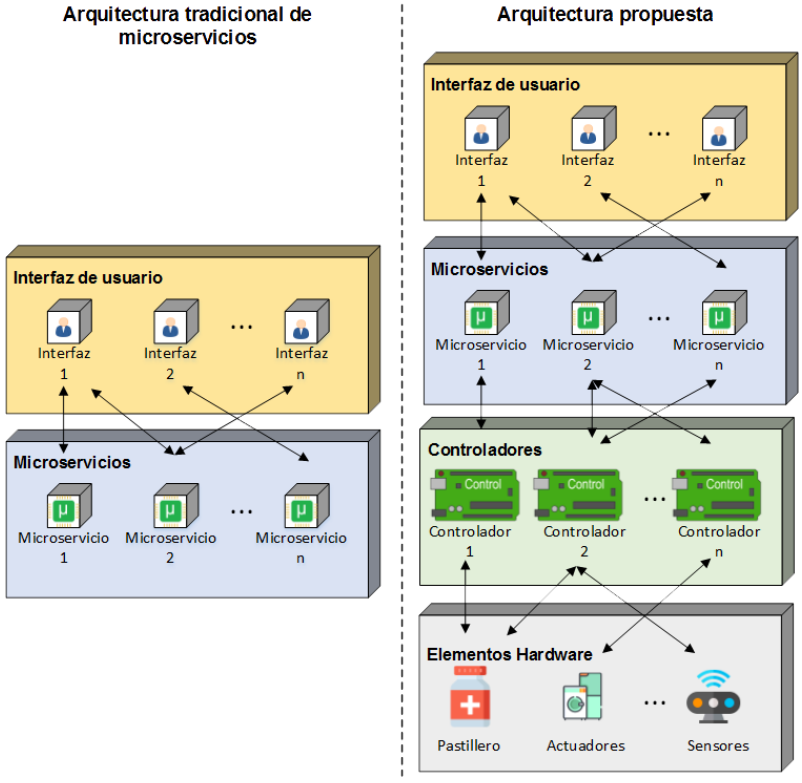


Figura 4.13: Comparación entre arquitectura general de aplicaciones de microservicios y arquitectura propuesta para IoT (Fuente: Elaboración propia).

A continuación, se describe cada una de las capas de la arquitectura recomendada por MicroIoT, pero se enfatiza en el hecho de que la metodología no está estrictamente acoplada con esta arquitectura, lo que resulta ventajoso dado que puede ser reemplazada con una arquitectura que se apegue incluso a otros dominios.

I **Interfaz de usuario:** Esta capa se encuentran las vistas ofrecidas a los usuarios finales, es una representación visual de las diferentes interfaces (página web, aplicación de escritorio, o aplicación móvil, entre otras) a través de las cuales un usuario puede acceder a los servicios ofrecidos por uno o varios microservicios.

II **Microservicios:** Esta capa contiene los microservicios identificados en la

actividad de Diseño de la entrega Sección 4.4.3, se debe especificar, cómo un microservicio se vincula con determinadas interfaces de usuario y en el caso de IoT, con un determinado grupo de controladores, según corresponda a los identificados por el Analista de requerimientos y los Interesados, en la Sección 4.4.3.

III **Controladores:** Esta capa almacena los diferentes controladores requeridos en la entrega. Un controlador se relaciona con un microservicio si juntos cumplen con un requerimiento de la entrega. Los controladores más destacados son: Raspberry Pi, Arduino, con módulo wifi o cualquier controlador genérico con acceso a internet para poder comunicarse con los servicios que ofrece cada microservicio.

IV **Elementos de Hardware:** Esta capa contiene elementos tales como: sensores, actuadores que conforman el entorno IoT. Para que un elemento de hardware pueda formar parte de un entorno IoT, es necesario que se encuentre vinculado a un controlador.

En el caso de dispositivos con controladores y sensores integrados tales como: *smartphones, tablets*, computadores, electrodomésticos inteligentes, *smart tvs*, entre muchos otros, estos son considerados controladores, dado que cumplen sus funcionalidades y tienen conexión a internet, y a la vez cuentan con elementos de hardware, que cumplen las funciones de sensores o actuadores, estos pueden ser considerados con sistemas externos de IoT, como se definió en la tarea de identificación y verificación de microservicios en la Sección 4.4.3.

B. Establecimiento de la arquitectura de gestión de microservicios

Luego de contar con la arquitectura de la aplicación, es necesario establecer la arquitectura de gestión de microservicios, la cual abarca aspectos tales como el estilo de gestión de datos, estilo de comunicación entre microservicios, estilo de despliegue, presentación de microservicios entre otros. Los patrones existentes son presentados en la Figura 4.14 y fueron explicados en la Sección 2.1.3.A.

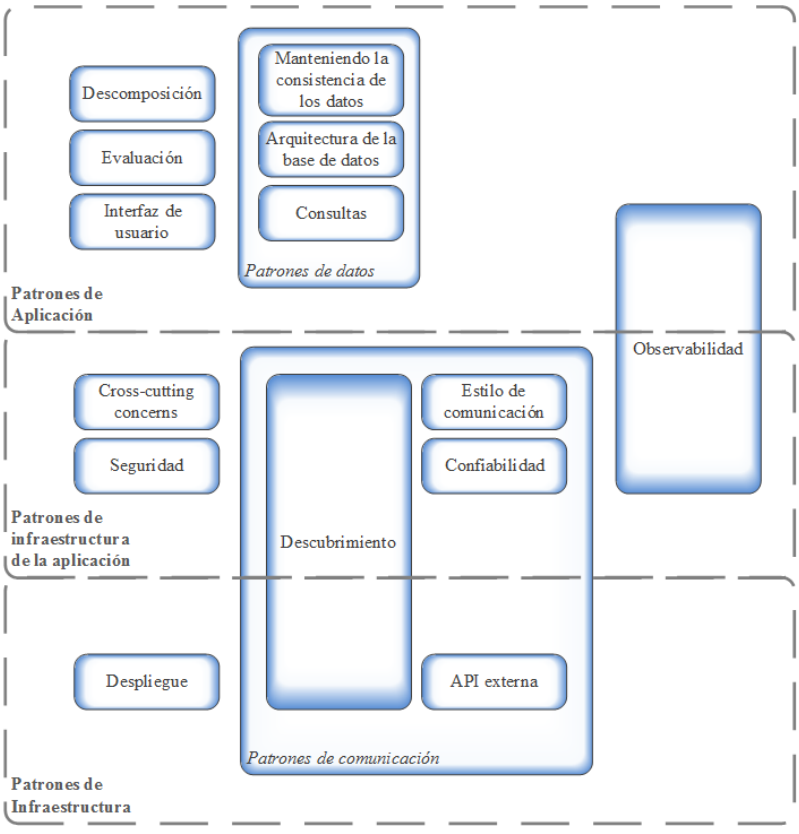


Figura 4.14: Patrones de gestión de microservicios. (Fuente: (Richardson, 2014))

Para la adopción de la Arquitectura de Microservicios en Internet de las Cosas se deben tomar en cuenta las siguientes consideraciones:

I **Acceso a microservicios**

El acceso a los microservicios se da por medio de APIs públicas, expuestas por servicios que aplican el patrón *"Remote Procedure Invocation"*, con base en los protocolos SOAP y REST (Richardson, 2014). De entre estos patrones, el estilo arquitectónico RESTful que usa REST y HTTP sobre TCP es particularmente atractivo para conectar dispositivos de premisas de consumo, dada la disponibilidad casi universal de pilas HTTP para varias plataformas (Vresk and Čavrak, 2016).

II **Monitoreo y prevención de fallas en cascada**

Para tareas de monitoreo, cada servicio debe proporcionar una interfaz que permita entregar información acerca del estado de los microservicios, es decir; el estado del microservicio, que microservicio se encuentra activo, y cuál ha tenido una falla. Es por ello que Butzin et al. (2016) nos recomienda tres patrones a adoptar al momento de desarrollar los microservicios.

- En primer lugar, la adopción del patrón Circuit Breaker (perteneciente a los tipos de patrones de confiabilidad del diseño de microservicios) el cual comprueba el estado de un servicio, así como el número de llamadas y peticiones fallidas y lo notifica en caso de superar un determinado umbral. Además de verificar el estado de los servicios, este patrón evita enviar mensajes innecesarios a servicios caídos reduciendo el tráfico en la red.
- De igual manera, Butzin et al. (2016), recomienda la adopción del patrón de cloud computing "Load Balancer" con el que se distribuye el trabajo entre un conjunto de servicios iguales, debido a que en IoT, el balanceador de carga puede aumentar la vida útil de los sensores inalámbricos ya que la carga de trabajo se comparte entre varios dispositivos, lo que les permite permanecer más tiempo en modos de baja potencia.

Al trabajar estos patrones en conjunto, el patrón *Circuit Breaker* permite que el equilibrador de carga trabaje solamente en servicios que se encuentran operativos eliminando el procesamiento de servicios caídos. En IoT, estos patrones se pueden usar solos o combinados. Ambos patrones han demostrado ser una buena manera de manejar la falla de los servicios remotos. Con respecto a la naturaleza limitada de muchos dispositivos de IoT, estos patrones tienen beneficios adicionales:

1. El balanceador de carga puede aumentar la vida útil de los sensores inalámbricos ya que la carga de trabajo se comparte entre varios dispositivos, lo que les permite permanecer más tiempo en modos de baja potencia.
2. Como el patrón *Circuit Breaker* puede ser utilizado por cada servicio (según sea necesario) sin tener en cuenta el servicio llamado, este patrón siempre es posible, incluso si el servicio llamado es provisto por otro proveedor. Por lo tanto, esta es una buena forma de proporcionar resistencia a las aplicaciones de IoT.

Relacionado con la detección y prevención de fallas Butzin et al. (2016), recomienda el patrón "Service registry" perteneciente a los patrones de descubrimiento de servicios. Este patrón recomienda utilizar un formato de registro en todos los servicios. Ésto ayuda a agregar registros individuales para obtener una imagen completa del sistema en general. En IoT, sin embargo, esto no es adecuado ya que probablemente nadie tenga el control de todos los servicios alojados en un escenario de IoT. Sin embargo, los desarrolladores de los servicios de IoT deben enfatizar el uso de un formato de registro común, por ejemplo: con marcas de tiempo internacionalizadas para facilitar la integración con otros registros.

Cuando se implementan dos capas (capa de interfaces de usuario y capa de controladores) éstas trabajan como el patrón *back-end for front-end* descrito por Richardson (2014) de tipo "API externa" debido a que se genera un API gateway diferente por cada tipo de interesado.

4.4.5. Validación del diseño y arquitectura de la entrega

Esta actividad es una de las más importantes de todo el proceso en general. Aquí el Arquitecto de Software junto con el Equipo Principal presentan el diseño obtenido a los Interesados, indicando los servicios que cumplen con los requerimientos, el principal objetivo es validar si la entrega diseñada corresponde a las expectativas del usuario y si el problema de software que abarca la entrega ha sido entendido adecuadamente y la solución propuesta cumple con los deseos de los interesados.

Para un entorno de Internet de las cosas, adicionalmente, se deberá validar con los interesados, las diferentes vistas de la entrega, es decir, las interfaces gráficas a las que cada usuario final tendrá acceso. También se deben tomar en cuenta los dispositivos de Internet de las Cosas disponibles o necesarios como los controladores, sensores y actuadores considerados en el diseño de la solución, esto debido al impacto que la adquisición o disponibilidad que estos recursos de hardware puedan tener en el coste del sistema; para ello, los interesados deberán contar con un experto de dominio para recomendar las mejores tecnologías de hardware. Al finalizar se debe contar con la aprobación de los interesados, la aplicación pasa a la fase de desarrollo.

Si el diseño no es aprobado por los interesados, se deberá identificar las fallas en la elicitación de requerimientos, y en las fases del diseño de la entrega. Luego de modo obligatorio, se debe regresar al inicio de la iteración reparando todos los errores cometidos, con especial cuidado de no repetirlos y no cometer nuevos errores.

4.4.6. Desarrollo y pruebas

Esta actividad implica tres prácticas de desarrollo de software que se presentan como tareas de esta actividad: i) desarrollo continuo, ii) evaluación continua, iii) integración continua, e intervienen los roles del equipo de desarrollo, equipo de aseguramiento de la calidad, y equipo de operaciones, es decir todo el equipo multifuncional trabaja en la tarea de desarrollo y pruebas. En la Figura 4.15 se ilustra el diagrama de procesos de esta actividad.

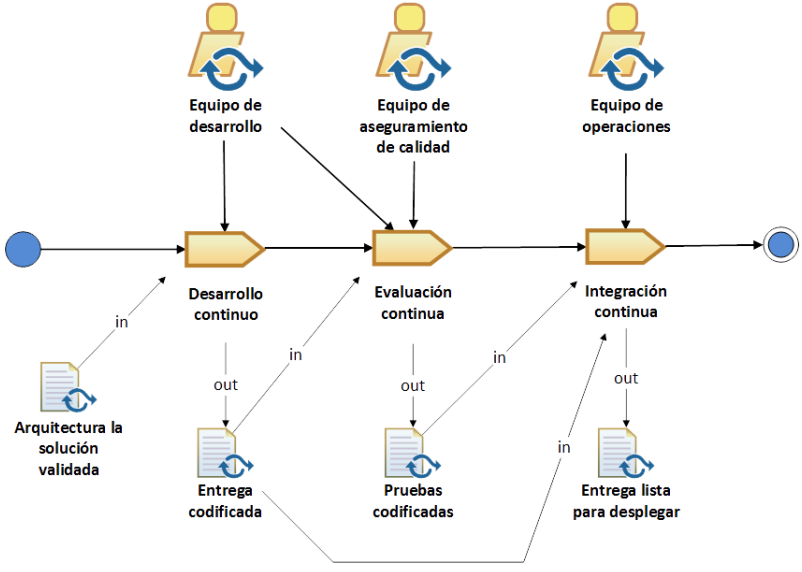


Figura 4.15: Fases de Actividad "Desarrollo y Pruebas" (Fuente: Elaboración propia).

Además es importante recordar que un microservicio incluye *front-end, back-end* y acceso a base de datos, y puede que el microservicio incluya funciones que no van a ser servicios en la nube, dado que se está implementando un sistema de IoT lo que implica la implementación de código en controladores o implementaciones para dispositivos móviles, en el caso de tener un front-end móvil o dispositivos móviles que hacen la veces de un controlador con sensores integrados.

A. Desarrollo continuo

En esta actividad cada equipo multifuncional se encarga de implementar o modificar un microservicio designado para que cumpla con determinadas funcionalidades requeridas. Para esta actividad, se emplean herramientas que faciliten la gestión y el versionamiento de código.

Luego de contar con el código desarrollado se requiere generar archivos ejecutables es por ello que se requiere herramientas que faciliten la gestión de dependencias y la compilación del código.

B. Evaluación continua

La evaluación continua implica realizar pruebas de manera anticipada y continua durante todo el ciclo de vida, esto resulta en costos reducidos, ciclos de prueba más cortos y una retroalimentación continua sobre la calidad. En esta fase el evaluador y el equipo de aseguramiento de calidad elaboran las pruebas necesarias para comprobar que los servicios del microservicio funcionan correctamente.

C. Integración continua (CI)

Como lo propone Fowler and Foemmel (2006), la integración continua es una práctica de desarrollo de software donde los miembros de un equipo integran su trabajo con frecuencia, lo que lleva a integraciones múltiples por día. Cada integración de código se verifica mediante una compilación automatizada (incluidas las pruebas de calidad) para detectar errores de integración lo más rápido posible. Este enfoque conduce a problemas de integración significativamente reducidos y permite a un equipo desarrollar software cohesivo más rápidamente. En MicroIoT sugerimos el uso de herramientas que permitan la automatización de esta tarea, integrando el trabajo de los desarrolladores tan pronto como sea posible, de esta manera, el sistema se prueba constantemente. La tarea de Integración continua incluye las siguientes tareas:

- Comprobación del código en el repositorio.
- Ejecución de pruebas unitarias.
- Las nuevas funcionalidades se integran con las preexistentes para buscar algún problema de integración.

Al superar las pruebas de integración la aplicación está lista para ser desplegada en el entorno de producción.

De acuerdo con Ebert et al. (2016), en el entorno de IoT el uso de CI no siempre es fácil y directo, las pruebas pueden significar un reto porque la construcción y las pruebas en el hardware de destino no siempre son posibles y, por lo tanto, deben simularse. Además, las limitaciones del hardware afectan la velocidad de prueba y, por lo tanto, el tiempo necesario para esta tarea.

4.4.7. Despliegue

El objetivo perseguido en esta fase es permitir lanzar nuevas funcionalidades a los clientes y usuarios lo más pronto posible, reduciendo la intervención humana. Durante esta fase se deben considerar dos tipos de herramientas involucradas al entorno de despliegue:

- Herramientas que permitan gestionar el entorno, realizar y superar pruebas relacionadas al entorno (como pueden ser pruebas de rendimiento, funcionalidad o seguridad)
- Herramientas para la gestionar la configuración de aplicaciones en un entorno de despliegue.

4.4.8. Operaciones

Esta actividad está alineada a la fase de operaciones de DevOps, la cual incluye dos prácticas, que permiten: i) monitorear cómo las aplicaciones lanzadas se están desempeñando en la producción y ii) recibir retroalimentación de los clientes. Esta información permite a las empresas reaccionar de manera ágil y cambiar sus planes comerciales según sea necesario (Sharma and Coyne, 2013). El diagrama de procesos de esta actividad se presenta en la Figura 4.16.

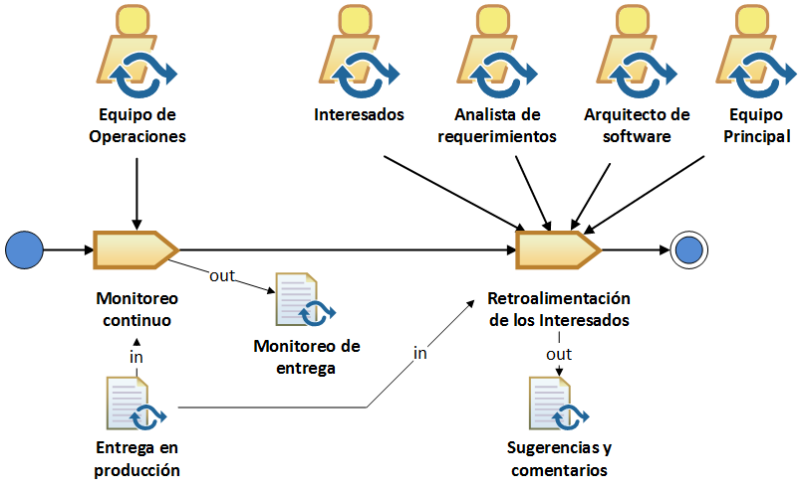


Figura 4.16: Fases de Actividad "Operaciones" (Fuente: Elaboración propia).

A. Monitoreo continuo

El monitoreo continuo proporciona datos y métricas a las operaciones, al control de calidad, al personal de desarrollo de líneas de negocio y a otras partes interesadas, acerca de las aplicaciones en las diferentes etapas del ciclo de entrega, el monitoreo es una tarea automatizada mediante herramientas.

I Herramientas de monitoreo

Permiten que las organizaciones identifiquen y resuelvan los problemas de la infraestructura de TI (Tecnologías de la Información) antes de que afecten a los procesos comerciales críticos. Estas herramientas supervisan aspectos del sistema tales como la carga de la CPU, la asignación de RAM, las estadísticas de tráfico de red, el consumo de memoria y la disponibilidad de espacio libre en el disco (Ebert et al., 2016).

B. Retroalimentación de los Interesados

Al estar en un entorno de producción, el sistema de software puede ser usado por los interesados y usuarios finales, en caso de surgir peticiones de cambios o nuevos requerimientos por parte de los interesados, también se deberá incurrir en una nueva iteración de la metodología, pero con un enfoque especial en la priorización de requerimientos para definir adecuadamente las

entregas pendientes y el orden en el que serán puestas en producción para uso de los interesados y usuarios finales.

En caso de haber entregas pendientes, se procede a realizar una nueva iteración de la metodología MicroIoT, hasta cubrir todas las entregas. Si además de entregas pendientes surgieron nuevos requerimientos, se deben analizar los nuevos requerimientos y establecer el orden de prioridad junto con las entregas pendientes para que el Arquitecto de Software pueda redefinir las entregas.

Edwin Cabrera Alvarado
Paola Cárdenas Cárdenas

Capítulo 5

Instanciación de la metodología

Este capítulo muestra una implementación de un sistema de software que instancia MicroIoT, la metodología propuesta en el capítulo 4, llevando a la vida práctica cada una de las actividades que intervienen en la misma. Las siguientes secciones muestran la ejecución estricta y lineal de cada fase de la metodología, en la Figura 4.2 se ilustraron las actividades que rigen la metodología MicroIoT, dicha figura se repite en la Figura 5.1, dado que esas mismas actividades guiarán el desarrollo del presente capítulo.

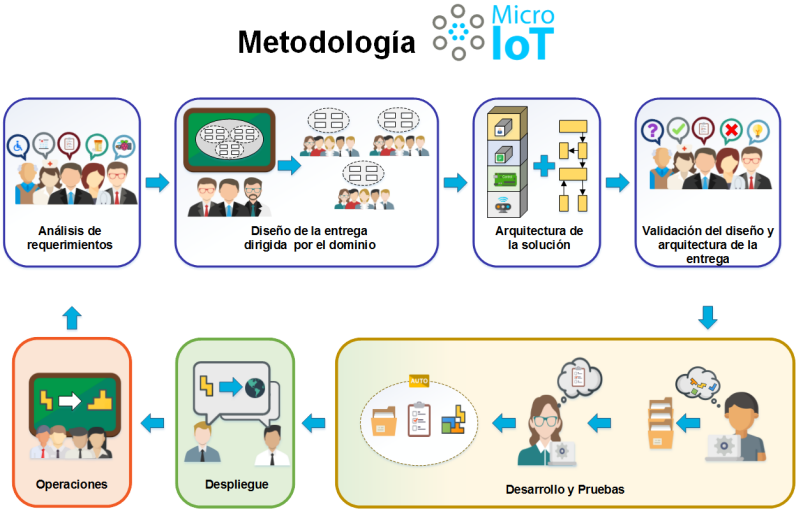


Figura 5.1: Actividades de MicroIoT (Fuente: Elaboración propia)

A continuación se presenta la ejecución de MicroIoT en un escenario práctico de la vida real. Considerando las actividades: i) Análisis de requerimientos, ii) Diseño de la entrega dirigida por el dominio, iii) Arquitectura de la solución, iv) Validación del diseño y arquitectura de la entrega, v) Desarrollo y Pruebas, vi) Despliegue y vii) Operaciones.

5.1. Definición de roles

Cada autor de MicroIoT asumirá múltiples roles, necesarios para instanciar el sistema de software que sigue estrictamente las actividades propuestas en la metodología. La Tabla 5.1 muestra los roles asumidos por cada autor.

Autor	Rol	Tareas en la que interviene
Autor 1	**Interesados**	• Elicitación de requerimientos. • Priorización de requerimientos. • Análisis del dominio.
Autor 2	**Analista de requerimientos**	• Elicitación de requerimientos. • Priorización de requerimientos. • Análisis del dominio. • Delimitación de contextos.
	Arquitecto de software	• Priorización de requerimientos. • Análisis del dominio. • Delimitación de contextos. • Definición de entidades agregados y servicios. • Identificación y verificación de microservicios. • Constitución de equipos de desarrollo.
Autor 1 + Autor 2	**Equipo Principal**	• (Se mantiene en comunicación constante con el arquitecto de software).

Tabla 5.1: Roles de usuario y responsabilidades identificadas

5.2. Análisis de requerimientos

MicroIoT plantea el análisis de requerimientos como la primera actividad, en la primera iteración de la metodología se definen las entregas en base a una elicitación de requerimientos de todo el sistema de software de IoT para AAL, luego se procede a realizar una priorización de requerimientos en el que serán desarrolladas. La Figura 5.2 muestra las tareas a realizar en esta actividad.

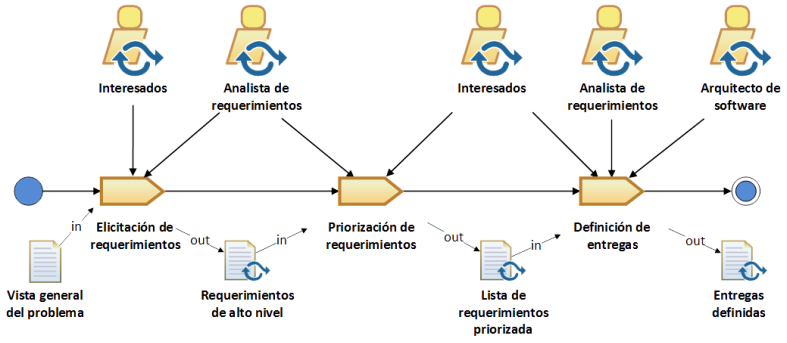


Figura 5.2: Tareas de la actividad: "Análisis de Requerimientos"

5.2.1. Elicitación de requerimientos

Como lo describe MicroIoT se realiza el proceso de elicitación de requerimientos para obtener una vista general del problema o solución de IoT para AAL e identificar los entregables requeridos.

A. Escenario planteado

Se desea un sistema de Internet de las Cosas para un ambiente de vida asistido, el objetivo es que los pacientes puedan ser monitoreados por su doctor, el doctor podrá consultar la frecuencia cardiaca del paciente y controlar el número de pasos que el paciente dió durante el día, además se quiere brindar a los pacientes la posibilidad de controlar la iluminación de su hogar sin hacer mayor esfuerzo.

B. Entregables del sistema (Requerimientos de alto nivel)

Los interesados manifiestan que desea obtener lo siguiente:

- Una plataforma web para que el doctor pueda ingresar y consultar la información de sus pacientes.
- Un dispositivo móvil capaz de detectar la frecuencia cardiaca y los pasos que da el paciente durante el día.
- Una aplicación que permita al paciente controlar la iluminación de su hogar, sin tener que hacer mayor esfuerzo, se requieren al menos dos maneras.

- Una alerta que indique el estado grave de los pacientes al doctor asignado doctor.

5.2.2. Priorización de requerimientos

El Analista de Requerimientos junto con los Interesados deciden priorizar los requerimientos por importancia, considerando por la urgencia de implementación dada la necesidad de los usuarios finales, a criterio del interesado, en donde el orden sería el siguiente:

1. Monitorear al paciente, lo cual involucra:
   a. Una plataforma web para que el doctor pueda ingresar y consultar la información de sus pacientes.
   b. Una aplicación móvil capaz de detectar la frecuencia cardiaca y los pasos que da el paciente durante el día.
2. Ofrecer un ambiente asistido en el hogar del paciente, lo cual involucra lo siguiente:
   a. Una aplicación móvil que permita al paciente controlar la iluminación de su hogar, sin tener que hacer mayor esfuerzo.
3. Sistemas que ofrecen una mejor experiencia y funcionalidad al usuario final.
   a. Otro modo de funcionalidad para que el paciente controle la iluminación de su hogar, sin tener que hacer mayor esfuerzo.
   b. Una alerta que indique al doctor que su paciente se encuentra en un estado de salud grave.

5.2.3. Definición de entregas

El Analista de Requerimientos y los Interesados, junto con el Arquitecto de Software concluyen que la mejor manera de implementar el sistema es definir dos entregas y se acordó que cada entrega quede estimulada como sigue:

A. Primera entrega

En esta entrega se consideran los requerimientos con mayor prioridad, y es la entrega más larga debido a que esta puede definir la base del sistema de software de IoT para AAL que se pretende implementar:

I. Monitorear al paciente, lo cual involucra:

a. Una plataforma web para que el doctor pueda ingresar y consultar la información de sus pacientes.

b. Una aplicación móvil capaz de detectar la frecuencia cardiaca y los pasos que da el paciente durante el día.

II. Ofrecer un ambiente asistido en el hogar del paciente, lo cual involucra lo siguiente:

a. Una aplicación móvil que permita al paciente controlar la iluminación de su hogar, sin tener que hacer mayor esfuerzo.

B. Segunda entrega

La segunda entrega considera los requerimientos que tienen un menor orden de prioridad, y es más pequeña que la primera debido a que esta puede ser una integración o una implementación independiente, esto será analizado en una nueva iteración de la metodología MicroIoT.

I. Sistema de detección de señales y sistema de alerta de salud.

a. Otro modo de funcionalidad para que el paciente controle la iluminación de su hogar, sin tener que hacer mayor esfuerzo.

b. Una alerta que indique al doctor que su paciente se encuentra en un estado de salud grave.

A continuación se analiza y diseña la solución de software para IoT para AAL, para la primera entrega.

5.3. Diseño de la entrega

Al ser la primera iteración de MicroIoT, se considera la primera entrega para realizar la actividad de diseño de la entrega. A continuación se ejecutan las tareas de esta actividad, siguiendo los lineamientos planteados por MicroIoT que se ilustran en la Figura 5.3.

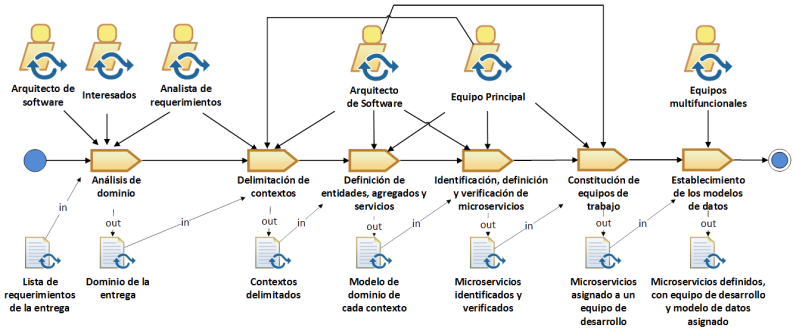


Figura 5.3: Fases de actividad "Proceso del Diseño de la entrega dirigida por el dominio"

5.3.1. Análisis del Dominio

En esta tarea MicroIoT recomienda identificar lo siguiente:

A. Identificación del problema

Implementar un sistema de IoT para AAL, el doctor puede consultar la frecuencia cardiaca y número de pasos que el paciente dio al día. Los pacientes pueden controlar la iluminación de su hogar, sin hacer mayor esfuerzo, por medio de una aplicación móvil.

B. Solución propuesta

La solución planteada por el arquitecto de software, misma que fue aprobada por el interesado consiste en lo siguiente:

- Crear una página web con login de usuario a la que pueden ingresar pacientes y doctores.
- Crear vistas independientes (páginas web) con los permisos y roles de usuario adecuados.
- Identificar las tecnologías adecuadas para el control de la iluminación del hogar y validar con el interesado para que se ajusten al presupuesto del proyecto.
- Crear una aplicación móvil que registre la frecuencia cardiaca y los pasos diarios que dio el paciente.

- Creación de un prototipo (maqueta), que muestre el funcionamiento del control de iluminación del hogar.

C. Requerimientos Generales de AAL

Se enfatizó en identificar los requerimientos que se muestran en la Tabla 5.2:

Requerimientos Generales de AAL Identificados	
Tipo o tipos de servicio de salud que abarca el sistema.	**Orientación del sistema (usuarios finales)**
Se identificaron dos tipos de servicio de salud: • Asistencia (para iluminación). • Vigilancia	El sistema está orientado a dos tipos de usuarios finales: • Paciente • Profesionales de la salud
Requerimientos de Hardware (Dispositivos de Internet de las Cosas disponibles o necesarios).	**Ambiente en el que opera el sistema**
En un acuerdo entre los involucrados, el analista de requerimientos y el arquitecto de software, se definió que se puede adquirir y/o disponer de los siguientes elementos de hardware: • Controlador raspberry pi 3 • Dispositivo móvil android 4.4.2 o superior, con cámara y un podómetro integrado (el paciente debe disponer del dispositivo móvil) • Focos (leds para el prototipo) • Cámara para la detección de señales manuales del paciente.	El sistema de IoT para AAL requerido, ofrece dos servicios diferentes que operan en diferentes lugares o ambientes. El servicio de asistencia para iluminación opera en el hogar del paciente, mientras que el servicio de vigilancia puede ser fácilmente implementado en las habitaciones de un hospital o ser ubicuo, pero en esta ocasión será el hogar del paciente. • Hogar • *Ubiquitous Health*
Plataforma de despliegue de la o las aplicaciones del sistema: Para la primera entrega se identificó la necesidad de construir el sistema de software IoT para AAL requerido, en aplicaciones independientes: • **Aplicación móvil** para el paciente para registrar los pasos que dio durante el día y la frecuencia cardiaca. • **Aplicación web** a la que accede el doctor para ver el estado de sus pacientes, a esta página web también tiene acceso el paciente para consultar y modificar el estado de la iluminación de su hogar. • **Aplicación de escritorio** como otra opción para el usuario para consultar y modificar el estado de la iluminación de su hogar.	

Tabla 5.2: Características Generales de AAL

D. Requerimientos de Calidad

Los requerimientos no funcionales identificados se muestran en la Tabla 5.3.

Aspectos de Calidad de Software Identificados.	
Aspectos de calidad según ISO 25010.	**Estándares de Salud**
Tanto el analista de requerimientos, el arquitecto de software han considerado importantes los siguientes aspectos: • Seguridad. • Usabilidad. • Interoperabilidad. • Eficiencia. • Disponibilidad. • Mantenibilidad. • Autenticidad.	El interesado junto con el arquitecto de software acordaron usar el conjunto de estándares de salud *Health Level Seven* (HL7), para el intercambio de información. • HL7 *Health Level Seven*

Tabla 5.3: Aspectos de Calidad de Software

5.3.2. Delimitación de contextos

De acuerdo a la metodología MicroIoT, la delimitación del contexto está a cargo del Analista de Requerimientos, del Arquitecto de Software y el Equipo Principal. Quienes prestan atención a dos características de las que se vieron en la tarea anterior: i) Tipos de servicio de salud y ii) Ambientes en los que opera el sistema.

Se identificaron dos tipos de servicio de salud:

- Asistencia (para iluminación)
- Vigilancia

Los ambientes en los que opera el sistema son los siguientes:

- Hogar
- Ubiquitous Health

De estos se decide que los términos que mejor describen los contextos son los tipos de servicio de salud, debido a que los requerimientos describen la necesidad de vigilancia y asistencia para el paciente. En la Figura 5.4 se muestra la delimitación de contextos, el nombre del contexto se encuentra exterior al contexto representado por la línea punteada, mientras que el interior se encuentra una breve descripción del subdominio del sistema que se pretende construir.

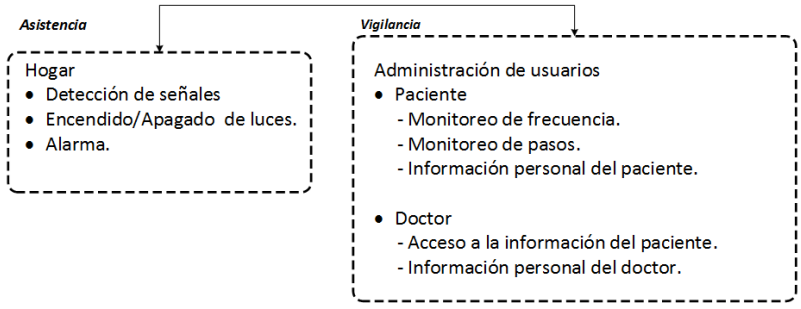


Figura 5.4: Delimitación de contextos

5.3.3. Definición de entidades, agregados y servicios

En cada contexto existe un subdominio, para ello se debe definir claramente el modelo de dominio, el cual está compuesto por entidades, objetos de valor que se presentan como atributos en este caso, agregados, servicios que vienen siendo las funciones de cada una de las entidades.

A. Identificación de Entidades

Después de realizar una identificación de sustantivos en los requerimientos funcionales identificados, además se consideró: i) La orientación del sistema que va dirigido a pacientes y doctores, ii) El ambiente en el que se opera el sistema que es el hogar, iii) Dispositivos de HW que son controladores y actuadores. Identificando las siguientes entidades:

- Entidad Persona (Doctores y pacientes son personas, por lo tanto comparten atributos comunes, por lo que se consideró la entidad persona)
  - Entidad Paciente
  - Entidad Doctor
- Entidad DispositivoHW
- Entidad ControladorHogar

B. Identificación de Objetos de Valor

Algunos de los atributos de los pacientes llegan a ser objetos de valor cuando se los presenta como una entidad sin identificador único, en este caso se identifica a “DatosSalud” como un obeto de valor de la entidad paciente.

C. Identificación de Agregados

Dado que los agregados están dados por las entidades raíces o entidades de las cuales otras entidades son dependendientes, las siguientes han sido identificadas como entidades raíz de los agregados.

- Persona
- DispositivoHW

Una vez que se han identificado los componentes necesarios para el diseño de la entrega dirigido por el dominio, se procede a definir los submodelos de dominio de cada contexto delimitado y la conexión existente entre cada contexto. En la Figura 5.5 se muestra el modelo de dominio junto con la delimitación de contextos.

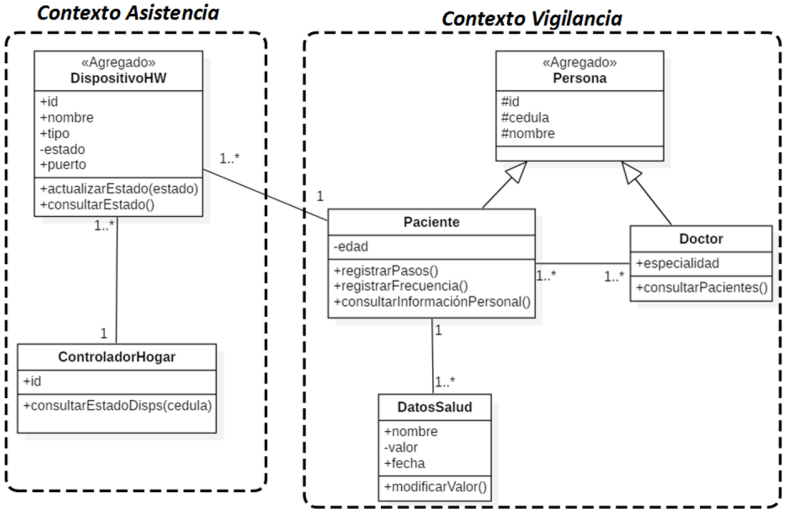


Figura 5.5: Modelo de dominio con contextos delimitados, entidades, objetos de valor y agregados agregado de DDD

La siguiente tarea es la identificación y verificación de microservicios, el modelo de dominio, los contextos delimitados y los agregados ayudaron a identificar con claridad los microservicios necesarios para la implementación del sistema de software de IoT para AAL, como se describe a continuación.

5.3.4. Identificación y verificación de microservicios

El Arquitecto de software junto con el Equipo Principal realizan la tarea de identificación y verificación de microservicios, basándose en las recomendaciones de MicroIoT para ésta tarea.

A. Identificación de microservicios

Basado en el principio básico planteado en MicroIoT, que estipula lo siguiente:

- Un microservicio no debe ser menor que un agregado ni mayor que un contexto delimitado.
- Los agregados suelen ser buenos candidatos para microservicios.

Se identifican los siguientes microservicios:

I. **Microservicio Hogar**

   El microservicio Hogar se encarga de proporcionar los servicios para el contexto de asistencia de iluminación, así como de ofrecer una interfaz gráfica para el control y consulta del estado de la iluminación del hogar.

II. **Microservicio Persona**

   Este microservicio se encargará del manejo la información de los usuarios, para proporcionar el servicio de *login*, además de los servicios de registro de frecuencia cardiaca y pasos que dio el paciente durante el día, y la gestión de la información personal de pacientes y doctores. Los usuarios finales (pacientes y doctores) podrán acceder a su información personal y en el caso del doctor, adicionalmente podrá acceder a la información de sus pacientes, por medio de una interfaz de usuario web. El microservicio persona se relaciona con el microservicio hogar dado que cada hogar le pertenece a un paciente.

La Figura 5.6 muestra los microservicios identificados en esta tarea, mediante una simbología que ilustra los microservicios vistos como aplicaciones pequeñas con sus respectivos componentes.

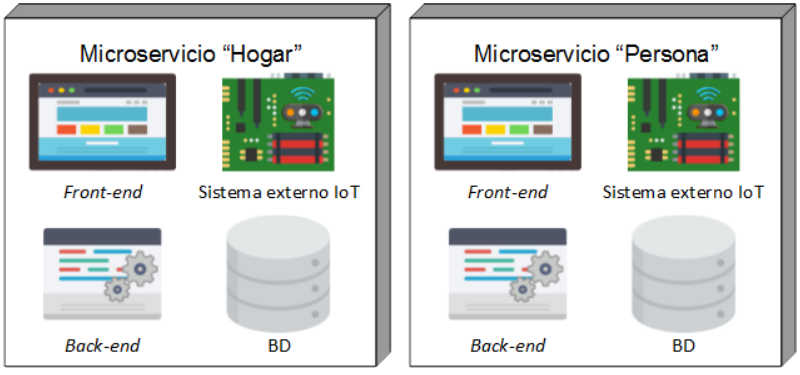


Figura 5.6: Microservicios Identificados con sus respectivos componentes (Fuente: Elaboración propia).

Los microservicios identificados deben ser verificados para tener mayor seguridad de que la solución basada en microservicios está siendo diseñada de manera correcta.

B. Verificación de microservicios

Para verificar si los microservicios han sido correctamente identificados, el Arquitecto de software junto con el Equipo Principal, aplican los criterios recomendados en MicroIoT, como se muestra en la Tabla 5.4.

Criterios de verificación de microservicios	Microservicios Identificados (Hogar, Persona)
a. Cada microservicio tiene una única responsabilidad y se conforma de *front-end*, *back-end* y su lógica de negocio.	✓
b. No hay llamadas que generen una alta conexión y dependencia entre microservicios. Si dividir la funcionalidad en dos microservicios hace que estos generen demasiada conversación, esto puede ser una indicación de que estas funciones deben estar en el mismo microservicio.	✓
c. Cada microservicio es lo suficientemente pequeño como para que un equipo pequeño lo pueda generar trabajando de forma independiente.	✓
d. No hay interdependencias que requieran la implementación de dos o más microservicios en sincronía. Siempre debe ser posible implementar un microservicio sin tener que volver a implementar ninguno de los demás microservicios.	✓
e. Los microservicios no están estrechamente acoplados y pueden evolucionar independientemente.	✓
f. Los límites del microservicio no causarán problemas con la coherencia de datos o la integridad.	✓
g. Todas las entidades aparecen exactamente una vez en el esquema. Otras entidades pueden contener referencias a él, pero no lo duplican.	✓

Tabla 5.4: Verificación de microservicios, aplicando las recomendaciones de MicroIoT

Una vez que los microservicios se han identificado plenamente y además han sido verificados, se procede a constituir los equipos de trabajo o equipos multifuncionales, recordando que un equipo de trabajo puede encargarse de

más de un microservicio.

5.3.5. Constitución de equipos de trabajo

En vista de que se cuenta con dos personas (los autores de MicroIoT), el Arquitecto de Software, decide que cada autor representará un equipo multifuncional y cada equipo multifuncional se encargará de un microservicio, además cada equipo multifuncional cuenta con un integrante del equipo principal, los cual ha estado trabajando junto con el Arquitecto de Software durante la tarea de diseño de la primera entrega. Esta decisión se basó en las recomendaciones de MicroIoT, la cuales estipulan lo siguiente:

"Un equipo de trabajo debe encargarse de uno o más microservicios, el encargado de asignar el o los microservicios a cada equipo de trabajo es el Arquitecto de Software, se recomienda que los microservicios asignados a un mismo equipo de trabajo pertenezcan a un mismo contexto o contextos parecidos para que el equipo se sienta más familiarizado con el lenguaje ubico de cada microservicio"

En la Figura 5.7 se ilustra la asignación de los microservicios a cada equipo de trabajo, conocido como equipo multifuncional, establecido por el Arquitecto de Software.

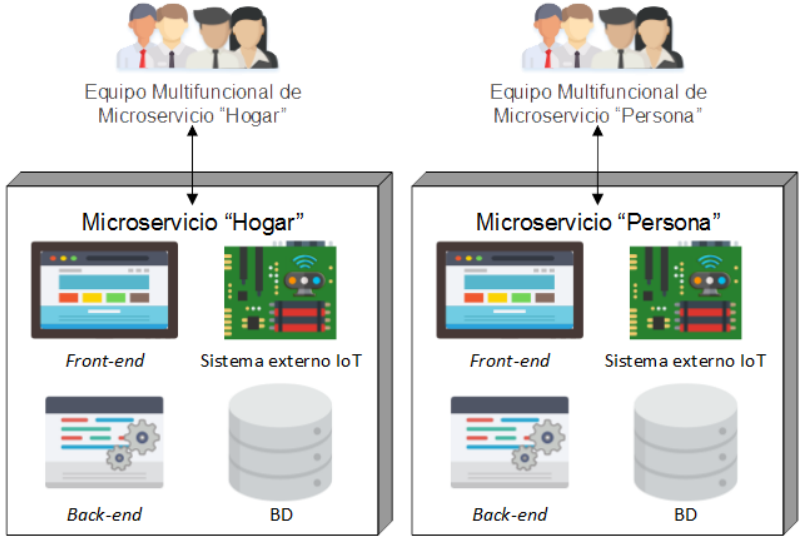


Figura 5.7: Asignación de microservicios a equipos de trabajo (Fuente: Elaboración propia).

5.3.6. Establecimiento de los modelos de datos

Cada microservicio tiene su propio modelo de datos independiente de los otros microservicios, además se toma en cuenta que según MicroIoT, las entidades no deben repetirse en dos microservicios, solo se puede tener referencias a una entidad modelada en otro microservicio. Cada equipo multifuncional se encarga del modelo de datos de su respectivo microservicio asignado.

A. Modelo de Datos del Microservicio Persona

El Microservicio Persona se encarga de la gestión de información de los usuarios, tanto doctores como pacientes, además se encarga de registrar los datos de salud del paciente. Para el sistema de IoT para AAL que se pretende instanciar, los datos de salud son: i) Frecuencia cardiaca y ii) Pasos diarios que ha dado el paciente. Pero para ello se usa la entidad denominada "DatosSalud", la cual permite representar estos y otros datos de salud Si fuese necesario representarlos más adelante. En la Figura 5.8 se presenta el diagrama de base de datos correspondiente, modelado por el equipo multifuncional a cargo del Microservicio Persona.

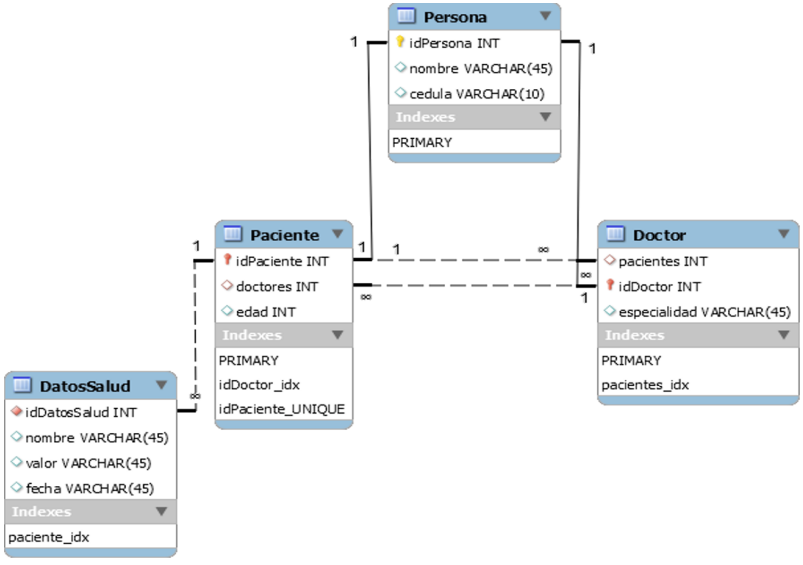


Figura 5.8: Diagrama de Base de Datos del Microservicio Persona (Fuente: Elaboración propia)

B. Modelo de Datos del Microservicio Hogar

El Microservicio Hogar se encarga de la gestión de dispositivos de *Hardware* y controladores de hogar, un dispositivo de *Hardware* está directamente asociado a un paciente, por ello se hace referencia a la entidad "Paciente" mediante el número de cédula; este atributo permite identificar a qué paciente le pertenece cada dispositivo de *Hardware*, mientras que los controladores son necesarios para exponer los servicios que serán ofrecidos por el Microservicio Hogar. En la Figura 5.9 se presenta el diagrama de base de datos correspondiente, modelado por el equipo multifuncional a cargo del Microservicio Hogar.

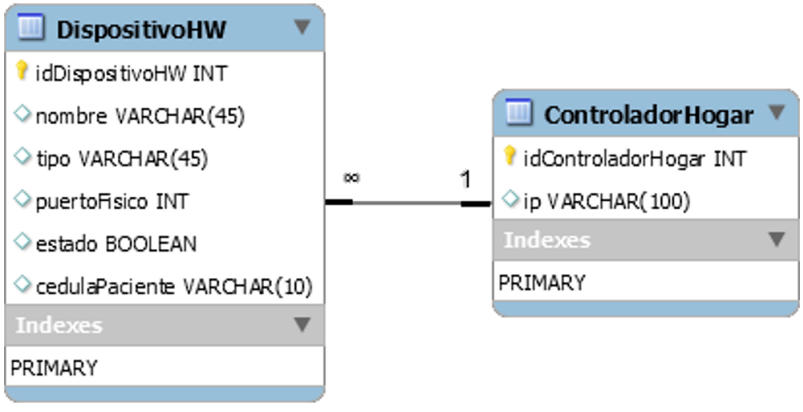


Figura 5.9: Diagrama de Base de Datos del Microservicio Hogar (Fuente: Elaboración propia)

5.4. Arquitectura de la solución

Como se menciona en el Capítulo 4, en esta actividad la solución se adapta a una a una Arquitectura de IoT basada en Microservicios, para definir los componentes de la Arquitectura, se tomaron en cuenta los requerimientos establecidos en las tablas 5.2 y 5.3.

5.4.1. Adecuación de la Arquitectura de la Aplicación

Al tener los microservicios claramente identificados, el Arquitecto de Software elabora la arquitectura descrita en la sección 4.4.4.A. Para la aplicación la arquitectura resultante se presenta en la Figura 5.10.

A. Interfaz de usuario

Para la interfaz de usuario se toma en cuenta la plataforma de despliegue de la Tabla5.2, por ello; para la interfaz de usuario se establece una página web la cual puede ser accedida Por los usuarios identificados en la tabla mencionada anteriormente (paciente y profesionales de la salud)

B. Microservicios

Los microservicios identificados en la sección 5.3.4.A A son aquellos que se ubican en esta capa de la arquitectura. En este caso, tanto "Microservicio Persona" como "Microservicio Hogar" se relacionan directamente con la página web mencionada anteriormente en la interfaz de usuario.

C. Controladores

En los controladores que forman parte de esta capa de la arquitectura se extraen de la Tabla 5.2 de la sección 5.3.1.C. En este caso se cuenta con raspberry pi para controlar las acciones relacionadas con micro servicio hogar y dispositivo Android se encarga de las tareas relacionadas con la gestión de microservicio paciente. Al ser el dispositivo Android un elemento que cuenta con sensores y actuadores su representación en esta capa de la arquitectura es un dispositivo Android con su componente de controlador

D. Elementos Hardware

De forma similar a la capa anterior de la arquitectura los elementos que forman parte de la capa de elementos de hardware se encuentran en la sección dispositivos de Hardware de la Tabla 5.2. Los elementos relacionados el control del hogar corresponden a los elementos de iluminación que se conecta de forma física con el Raspberry pi. Por otra parte, La cámara que desempeñará la función de un sensor de frecuencia cardíaca y el podómetro son los elementos de hardware integrados en un dispositivo Android los cuales interactúan con el controlador del mismo.

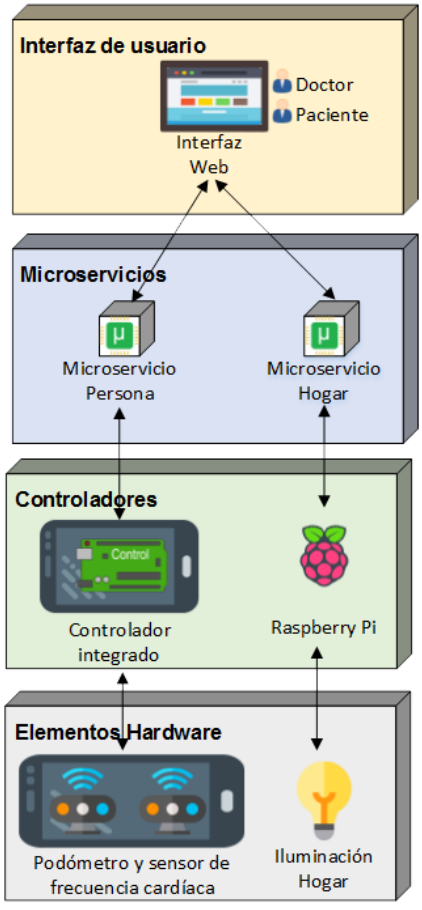


Figura 5.10: Diagrama de la arquitectura de la aplicación para la entrega 1

5.4.2. Establecimiento de la arquitectura de gestión de microservicios

Luego de tener la arquitectura de la aplicación es necesario establecer los patrones a implementar. En base a los patrones descritos por Richardson (2014) en la sección 2.1.3.A y a las recomendaciones descritas en el Capítulo 4, la arquitectura de gestión de microservicios contará con los patrones descritos en la Tabla 5.5.

Alineado a las capas	Categoría	Nombre Patrón
Patrón de aplicación	Patrones de descomposición	Descomposición por subdominios (se implementa como la actividad 2 de MicroIoT)
Patrón de aplicación	Patrones de datos (Arquitectura de base de datos)	Base de datos por servicio
Patrón de aplicación	Patrones de datos(Consultas)	API de Composición
Patrón de aplicación	Interfaz de usuario	Composición de fragmento de página de lado del servidor
Patrón de infraestructura de la aplicación	Seguridad	Acceso por Tokens
Patrón de infraestructura de la aplicación	Estilo de comunicación	Invocación de procedimientos remotos (REST)
Patrón de infraestructura de la aplicación	Confiabilidad	Circuit Breaker
Patrón de infraestructura	Despliegue	Servicios por contenedores
Patrón de infraestructura	Descubrimiento	Registro de servicio
Patrón de infraestructura	API externa	*Back-end* for *Front-end*

Tabla 5.5: Arquitectura de gestión de microservicios de la aplicación para la entrega 1

5.5. Validación del diseño y arquitectura de la entrega

La validación del diseño y arquitectura de la entrega es una de las actividades más importantes dentro de MicroIoT, por ello el Arquitecto de Software debe presentar la solución diseñada a los Interesados. Los criterios de validación aplicados se presentan en la Tabla 5.6.

Criterio de Validación	**Propuesta del Arquitecto de Software y Equipo Principal**	**Retroalimentación de los Interesados**
Entendimiento del Problema	Implementar un sistema de IoT para AAL, el doctor puede consultar la frecuencia cardiaca y número de pasos que el paciente dio al día. Los pacientes pueden controlar la iluminación de su hogar, sin hacer mayor esfuerzo, por medio de una aplicación móvil.	✓ Los interesados está de acuerdo con que esa es una vista de alto nivel del problema en cuestión y es lo que necesitan, obviamente se debe entrar en detalle para cada punto en cuestión.
Solución propuesta	La solución abarca una sola plataforma web en la que se cargará el contenido adecuado a cada usuario que ingrese a la página web. • Crear una página web con login de usuario a la que pueden ingresar pacientes y doctores. • Crear vistas independientes (páginas web) con los permisos y roles de usuario adecuados. • Crear una aplicación móvil que registre la frecuencia cardiaca y los pasos diarios que dio el paciente. • Creación de un prototipo (maqueta), que muestre el funcionamiento del control de iluminación del hogar	La solución planteada por el arquitecto de software, fue aprobada por el interesado durante el proceso de Análisis del Dominio. Sin embargo se vuelve a presentar la solución diseñada, a los Interesados para que confirmen nuevamente su aceptación. ✓ En este caso la respuesta fue satisfactoria, dado que los interesados están de acuerdo con la solución propuesta.
Usuarios finales	• Pacientes • Doctores	✓ Los interesados están de acuerdo.

Vistas o interfaces de usuario	• **Aplicación móvil** para el paciente para registrar los pasos que dio durante el día y la frecuencia cardiaca.	✓ Los interesados están de acuerdo.
	• **Aplicación web** a la que accede el doctor para ver el estado de sus pacientes, a esta página web también tiene acceso el paciente para consultar y modificar el estado de la iluminación de su hogar	✓ Los interesados están de acuerdo.
	• **Aplicación de escritorio** como otra opción para el usuario para consultar y modificar el estado de la iluminación de su hogar	✓ Los interesados están de acuerdo.
Ambiente en el que opera el sistema.	• **Hogar:** La iluminación está orientada al hogar del paciente. • ***Ubiquitous Health:*** El registro de los datos de salud del paciente pueden ser registrados de manera ubicua siempre y cuando el dispositivo móvil disponga de conexión a internet.	✓ Los interesados están de acuerdo.

Aspectos de calidad considerados	Tanto el Analista de Requerimientos, el Arquitecto de Software han considerado importantes los siguientes aspectos: • **Seguridad y Autenticidad.** Se considerará la autenticación de usuarios, usando el patrón *Access Token*, para el autenticacion de usuarios y manejo de de sesiones, además la contraseña será almacenada encriptada en la base de datos. • **Usabilidad.** Se consideran características de usabilidad como la facilidad y eficiencia de uso, además de la naturalidad de la interacción en los *front-end* de cada microservicios. • **Interoperabilidad.** Al usar microservicios que son agnósticos de las tecnologías, lenguajes de programación, y consideraciones internas de implementación, esto por ser independientes uno del otro, manejan su propio contexto como un entorno aislado.	✓ Los interesados están de acuerdo.

Aspectos de calidad considerados	• **Eficiencia.** Se proporciona eficiencia dado que el uso de recursos es adecuado, si bien se incluyen nuevos aspectos de la arquitectura de microservicios estos ayudarán a: i) una mejor distribución de los recursos, basada en la cantidad de peticiones y ii) una escalabilidad independiente para cada funcionalidad del sistema de IoT para AAL requerido. • **Disponibilidad.** Arquitectura de microservicios se acopla al uso de contenedores a partir de los cuales se pueden crear instancias de los microservicios, proveyendo asi disponibilidad constante. • **Mantenibilidad.** El hecho de que cada microservicio sea una aplicación independiente permite que los cambios posteriores y mantenibilidad en general sea también independiente, sin afectar a todo el sistema.	✓ Los interesados están de acuerdo.
Estándares de datos	• Para el registro de datos de salud de los pacientes, en este caso frecuencia cardiaca y cantidad de pasos diarios que dio el paciente, serán transmitidos en formatos bajo el estándar de datos HL7 *Health Level Seven.*	✓ Los interesados están de acuerdo.

Controladores necesarios	• Controlador raspberry pi 3 Será usado para controlar los focos del hogar del paciente. • Dispositivo móvil android 4.4.2 o superior, con cámara y un podómetro integrado (el paciente debe disponer del dispositivo móvil) Por medio del dispositivo móvil se registrará: i) la frecuencia cardíaca, usando la cámara integrada y ii) los pasos que dio el paciente durante en el día, por medio de un podómetro integrado en el dispositivo móvil.	✓ Los interesados están de acuerdo.
Elementos de Hardware necesarios	• Focos (leds para el prototipo) • Cámara para la detección de señales manuales del paciente.	✓ Los interesados están de acuerdo.
Arquitectura del sistema de software de IoT para AAL	Se puede usar el esquema de Arquitectura de Software de la Figura 5.9, que muestra claramente la interacción entre los elementos de hardware y controladores, además de las vistas del sistema de software, así como los usuarios finales o actores que tienen acceso a cada una de las vistas, omitiendo explicaciones técnicas que deben ser transparentes a los interesados, tales como las conexiones de los microservicios con el resto de componentes de la arquitectura.	✓ Los interesados están de acuerdo.

Tabla 5.6: Criterios de validación del diseño y arquitectura de la entrega.

5.6. Desarrollo y Pruebas

Cómo lo describe MicroIoT, cada equipo de trabajo se encarga de codificar El microservicio designado, tanto *back-end*, *front-end* y gestión de base de datos. A continuación se presenta el desarrollo que se siguió en cada uno de los microservicios.

5.6.1. Desarrollo y pruebas del Microservicio Persona

Debido a que el desarrollo de un microservicio implica codificación, manejo de versiones del código, compilación y como se recomienda en MicroIoT se debe automatizar, usar herramientas para automatizar el proceso. En la Tabla 5.7 se presenta las tecnologías utilizadas y la razón de la elección de dicha tecnología para él microservicio designado.

Tecnologías y herramientas	Herramienta	Función	Motivo de la elección
Framework	Spring Boot	Permite desarrollar aplicaciones que puedan ser desplegadas en la web facilitando la gestión de dependencias de las mismas.	Se seleccionó este framework debido a qué es el más utilizado y es aquél qué más soporte ofrece para la gestión de microservicios.
IDE	Spring Tool Suite	Permite desarrollar aplicaciones basadas en Spring.	Se utilizó debido a que proporciona un entorno para implementar ejecutar aplicaciones Spring.
WebAPI	Spring Initializer	Utilizado para construir el esqueleto de la aplicación.	Esta herramienta es utilizada debido a que permite generar aplicaciones web de forma rápida, Incluyendo el conjunto de dependencias que se requiera.

Herramienta	Maven	Se utiliza como gestor de dependencias Que serán requeridas para la implementación, gestión y despliegue del micro servicio.	Fue seleccionada esta herramienta debido a que es el gestor de dependencias por defecto de Spring.
Herramienta	Gradle	Se usó como manejador de dependencias, para la implementación de la aplicación móvil, que sirve de controlador de los sensores necesarios para la detección de datos de salud del paciente (frecuencia cardíaca y pasos diarios).	Es la herramienta usada por defecto en el entorno de Android Studio, en el que se implementó la aplicación móvil que hace el rol de controlador.
Framework	JUnit	Desarrollar pruebas unitarias de los servicios del microservicio.	Se utilizó debido A que es la herramienta más utilizada para la creación de pruebas unitarias en Java.
Sistema	GitHub	Permite gestionar todo código desarrollado, a través del versionamiento del mismo.	Es uno de los gestores de códigos más utilizados, a diferencia de Git, GitHub proporciona una interfaz web.
Plataforma	Docker	Facilita el despliegue de aplicaciones utilizando contenedores de software.	Es la herramienta de código libre más utilizada para despliegue de aplicaciones utilizando contenedores.

Servidor	Jenkins	Automatiza tareas relacionadas con la construcción evaluación y entrega de software.	Jenkins es un sistema de código abierto basado en Java y una de las herramientas más implementadas, por lo que las opciones de complemento son muchas. Debido a su popularidad, encontrar soporte de su gran base de usuarios es fácil (Ebert, 2016).

Tabla 5.7: Resumen de herramientas para desarrollo, evaluación, e integración para microservicio "Persona".

5.6.2. Desarrollo y pruebas del Microservicio Hogar.

En la Tabla 5.8 se presenta las tecnologías utilizadas para el desarrollo del Microservicio Hogar, así como la razón de la elección de dicha tecnología para él microservicio designado.

Tecnologías y herramientas	**Herramienta**	**Función**	**Motivo de la elección**
Framework	Spring Boot	Permite desarrollar aplicaciones que puedan ser desplegadas en la web facilitando la gestión de dependencias de las mismas.	Se seleccionó este framework debido a qué es el más utilizado y es aquél qué más soporte ofrece para la gestión de microservicios.
IDE	Spring Tool Suite	Permite desarrollar aplicaciones basadas en Spring.	Se utilizó debido a que proporciona un entorno para implementar ejecutar aplicaciones Spring.

WebAPI	Spring Initializer	Utilizado para construir el esqueleto de la aplicación.	Esta herramienta es utilizada debido a que permite generar aplicaciones web de forma rápida, Incluyendo el conjunto de dependencias que se requiera.
Herramienta	Maven	Se utiliza como gestor de dependencias que serán requeridas para la implementación, gestión y despliegue del microservicio.	Fue seleccionada esta herramienta debido a que es el gestor de dependencias por defecto de Spring.
Herramienta	Gradle	Se usó como manejador de dependencias, para la implementación de la aplicación móvil, que sirve de controlador de los sensores necesarios para la detección de datos de salud del paciente (frecuencia cardíaca y pasos diarios).	Es la herramienta usada por defecto en el entorno de Android Studio, en el que se implementó la aplicación móvil que hace el rol de controlador.
Framework	JUnit	Desarrollar pruebas unitarias de los servicios del micro servicio.	Se utilizó debido A que es la herramienta más utilizada para la creación de pruebas unitarias en Java.
Sistema	GitHub	Permite gestionar todo código desarrollado, a través del versionamiento del mismo.	Es uno de los gestores de códigos más utilizados, a diferencia de Git, GitHub proporciona una interfaz web.

Plataforma	Docker	Facilita el despliegue de aplicaciones utilizando contenedores de software.	Es la herramienta de código libre más utilizada para despliegue de aplicaciones utilizando contenedores.
Servidor	Jenkins	Automatiza tareas relacionadas con la construcción evaluación y entrega de software.	Jenkins es un sistema de código abierto basado en Java y una de las herramientas más implementadas, por lo que las opciones de complemento son muchas. Debido a su popularidad, encontrar soporte de su gran base de usuarios es fácil (Ebert et al., 2016).

Tabla 5.8: Resumen de herramientas para desarrollo, evaluación, e integración para microservicio "Hogar".

5.7. Despliegue

Como se menciona en la sección 4.4.7 las herramientas de despliegue tienen como objetivo gestionar el entorno donde se va a desplegar los microservicios así como la gestión de la configuración de los microservicios al ser desplegados en este entorno. Debido a que las herramientas de esta sección se desarrollan en un entorno real que acepta grandes cantidades de peticiones no se ha implementado este tipo de herramientas, sin embargo; existen herramientas que realizan pruebas sobre los microservicios para determinar su comportamiento en un entorno real, es por ello que en esta sección se aplicará las herramientas de la Tabla 5.9:

Tecnologías y herramientas	Herramienta	Función	Motivo de la elección
Herramienta	Turbine	Turbine es una herramienta para agregar flujos de datos JSON de eventos enviados por el servidor (SSE) a una única transmisión. Con esto se puede simular el comportamiento de un microservicio al recibir varias peticiones	Se seleccionó este framework debido a qué es el más utilizado y es aquél qué más soporte ofrece para la gestión de microservicios

Tabla 5.9: Resumen de herramientas aplicadas para despliegue de microservicios.

A pesar de no instanciar las herramientas de despliegue requeridas para la gestión de una infraestructura de DevOps en la Tabla 5.10 se presentan herramientas que podrían utilizarse en infraestructuras mucho más grandes.

Tecnologías y herramientas	Herramienta	Actividad involucrada	Descripción
Framework	Log4j	Logging	Log4j es una biblioteca open source que permite a los desarrolladores de software escribir mensajes de registro, cuyo propósito es dejar constancia de una determinada transacción en tiempo de ejecución
Sistema	Graylog	Logging	Graylog2 es un analizador de registro de código abierto que permite almacenar y buscar errores de registro.
Herramienta	Chef	Gestión de configuración	Chef es una herramienta de gestión de configuración utilizada para simplificar la tarea de configurar y mantener los servidores de una empresa.
Herramienta	Puppet	Gestión de configuración	Puppet es una utilidad de administración de configuración de código abierto. Se ejecuta en muchos sistemas tipo Unix, así como en Microsoft Windows, e incluye su propio lenguaje declarativo para describir la configuración del sistema.

Tabla 5.10: Resumen de herramientas recomendadas para despliegue de microservicios

5.8. Operaciones

5.8.1. Monitoreo continuo

Como se explica en MicroIoT, el monitoreo continuo permite que las organizaciones identifiquen y resuelvan los problemas de la infraestructura de TI (Tecnologías de la Información) antes de que afecten a los procesos comerciales críticos. Se trata de una tarea automatizada con herramientas, estas herramientas supervisan aspectos del sistema tales como la carga de la CPU, la asignación de RAM, las estadísticas de tráfico de red, el consumo de memoria y la disponibilidad de espacio libre en el disco. Al ser una aplicación pequeña y dado que los microservicios de la misma no tienen un gran número de peticiones, se decidió no instanciar este tipo de herramientas, dado que el aporte de la metodología MicroIoT no se centra en el monitoreo continuo y manejo de Tecnologías de la Información, sino en las actividades de diseño y arquitectura de la solución. Sin embargo; se instanció las herramientas esta tarea de la Tabla 5.11 ya que permite el monitoreo de microservicios de una manera sin considerar aspectos de la infraestructura.

Tecnologías y herramientas	Herramienta	Función	Motivo de la elección
Herramienta	Eureka	Eureka es un servicio REST (Transferencia de estado representacional) que se utiliza para el registro de microservicios	Se seleccionó este framework debido a qué es el más utilizado y es aquél qué más soporte ofrece para la gestión de microservicios

Tabla 5.11: Resumen de herramientas aplicadas en monitoreo continuo de microservicios

A pesar de no instanciar las herramientas especializadas en el monitoreo de la infraestructura. En la Tabla 5.12 se presentan herramientas que podrían utilizarse en infraestructuras mucho más grandes.

Tecnologías y herramientas	Herramienta	Actividad Involucrada	Descripción
Framework	Log4j	Logging	Log4j es una biblioteca open source que permite a los desarrolladores de software escribir mensajes de registro, cuyo propósito es dejar constancia de una determinada transacción en tiempo de ejecución.
Herramienta	Chef	Gestión de configuración	Chef es una herramienta de gestión de configuración utilizada para simplificar la tarea de configurar y mantener los servidores de una empresa.

Tabla 5.12: Resumen de herramientas aplicadas en monitoreo continuo de microservicios

5.8.2. Retroalimentación de los Interesados

Los interesados, han estado presentes durante la mayor parte del proceso aplicativo de MicroIoT, que tiene como objetivo construir un sistema de software IoT para AAL. Durante el ciclo de desarrollo de microIoT no han surgido nuevos requerimientos ni sugerencias.

A. Validación de la entrega

Luego de completar el primer ciclo de desarrollo de la metodología se obtienen los productos de software IoT, que cumple con todos los requerimientos especificados para esta entrega. En la validación de la primera entrega sistema de software de IoT para AAL, realizada en la sección 5.5, se usó una tabla de criterios de validación logrando la aprobación de los interesados. Ahora se valida la primera entrega codificada, verificada y desplegada, usando los mismos criterios establecidos en la validación del diseño y arquitectura de la entrega, en la Tabla 5.13 se presentan los criterios de validación establecidos para validar los resultados.

Criterio de Validación	**Propuesta del Arquitecto de Software y Equipo Principal**	**Retroalimentación de los Interesados**
Entendimiento del Problema	Implementar un sistema de IoT para AAL, el doctor puede consultar la frecuencia cardiaca y número de pasos que el paciente dio al día. Los pacientes pueden controlar la iluminación de su hogar, sin hacer mayor esfuerzo, por medio de una aplicación móvil.	✓ Los interesados está de acuerdo.

Solución propuesta	La solución abarca una sola plataforma web en la que se cargará el contenido adecuado a cada usuario que ingrese a la página web. También contempla una aplicación móvil Android. • Página web con login de usuario a la que pueden ingresar pacientes y doctores. En la Figura 5.10 se muestra la página web de login. • Vistas independientes (páginas web) con los permisos y roles de usuario adecuados. En la Figura 5.11 muestra una vista de la pantalla que verá el paciente y la Figura 5.12 muestra una vista de la pantalla que verá el doctor. • Crear una aplicación móvil que registre la frecuencia cardiaca y los pasos diarios que dio el paciente, en la Figura 5.13 se muestra la aplicación móvil para la detección y registro de datos de salud del paciente, la misma cuenta con un login de acceso para identificar al paciente. • Creación de un prototipo (maqueta), que muestre el funcionamiento del control de iluminación del hogar. En la Figura 5.14 se muestra la maqueta del hogar del paciente en la cual se controla la iluminación.	Con respecto a cada punto la retroalimentación de los interesados es la siguiente: ✓ De acuerdo. ✓ Los interesados están de acuerdo. Sin embargo en la Figura 5.12, se desea ver un estimado de los km que recorrió el paciente, además de los pasos que dió. ✓ De acuerdo. ✓ De acuerdo.
Usuarios finales	• Pacientes • Doctores	✓ Los interesados están de acuerdo.

Vistas o interfaces de usuario	• **Aplicación móvil** para el paciente para registrar los pasos que dio durante el día y la frecuencia cardiaca.	✓ Los interesados están de acuerdo.
	• **Aplicación web** a la que accede el doctor para ver el estado de sus pacientes, a esta página web también tiene acceso el paciente para consultar y modificar el estado de la iluminación de su hogar	✓ Los interesados están de acuerdo.
	• **Aplicación de escritorio** como otra opción para el usuario para consultar y modificar el estado de la iluminación de su hogar	✓ Los interesados están de acuerdo.
Ambiente en el que opera el sistema.	• **Hogar:** La iluminación está orientada al hogar del paciente. • ***Ubiquitous Health:*** El registro de los datos de salud del paciente pueden ser registrados de manera ubicua siempre y cuando el dispositivo móvil disponga de conexión a internet.	✓ Los interesados están de acuerdo.

Aspectos de calidad considerados	Tanto el Analista de Requerimientos, el Arquitecto de Software han considerado importantes los siguientes aspectos: • **Seguridad y Autenticidad.** Autenticación de usuarios, usando el patrón *Access Token*, para el autenticacion de usuarios y manejo de de sesiones, además la contraseña es almacenada encriptada en la base de datos. • **Usabilidad.** Se consideran características de usabilidad como la facilidad y eficiencia de uso, además de la naturalidad de la interacción en los *front-end*. • **Interoperabilidad.** El sistema de software de IoT para AAL es agnóstico de tecnologías. • **Eficiencia.** Se proporciona eficiencia dado que el uso de recursos es adecuado, si bien se incluyen nuevos aspectos de la arquitectura de microservicios estos ayudarán a: i) una mejor distribución de los recursos, basada en la cantidad de peticiones y ii) una escalabilidad independiente para cada funcionalidad del sistema de IoT para AAL requerido.	✓ Los interesados están de acuerdo.

Aspectos de calidad considerados	• **Disponibilidad.** Arquitectura de microservicios se acopla al uso de contenedores a partir de los cuales se pueden crear instancias de los microservicios, proveyendo asi disponibilidad constante. • **Mantenibilidad.** El hecho de que cada microservicio sea una aplicación independiente permite que los cambios posteriores y mantenibilidad en general sea también independiente, sin afectar a todo el sistema.	✓ Los interesados están de acuerdo.
Estándares de datos	• Para el registro de datos de salud de los pacientes, en este caso frecuencia cardiaca y cantidad de pasos diarios que dio el paciente, serán transmitidos en formatos bajo el estándar de datos HL7 *Health Level Seven. Health Level Seven.*	✓ Los interesados están de acuerdo.
Controladores necesarios	• Controlador raspberry pi 3: Usado para controlar los focos del hogar del paciente. • Dispositivo móvil android 4.4.2 o superior, con cámara y un podómetro integrado: Usado para el registro de los datos de salud del paciente.	✓ Los interesados están de acuerdo.
Elementos de Hardware necesarios	• Focos (leds para el prototipo) • Cámara para la detección de señales manuales del paciente.	✓ Los interesados están de acuerdo.

Arquitectura del sistema de software de IoT para AAL	La arquitectura está alineada a lo presentado en la Figura 5.9, que muestra claramente la interacción entre los elementos de hardware y controladores, además de las vistas del sistema de software, así como los usuarios finales o actores que tienen acceso a cada una de las vista.	✓ Los interesados están de acuerdo.

Tabla 5.13: Criterios de validación del diseño y arquitectura de la entrega.

En las Figuras 5.11,5.12,5.13 y 5.14 se presentan las interfaces de usuario pertenecientes, mientras que la Figura 5.15 presenta la maqueta de la del hogar del paciente.

Figura 5.11: Interfaz Gráfica del login de acceso a la página web

Figura 5.12: Vista de la interfaz gráfica que verán los pacientes

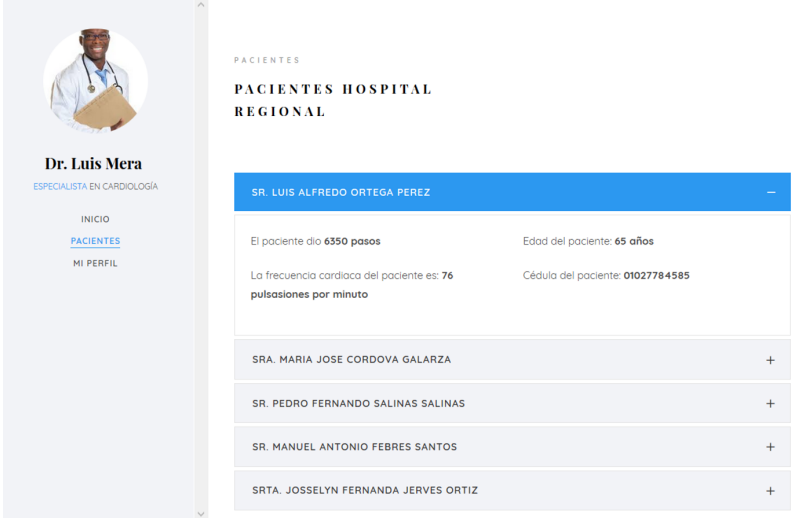


Figura 5.13: Vista de la interfaz gráfica que verán los doctores

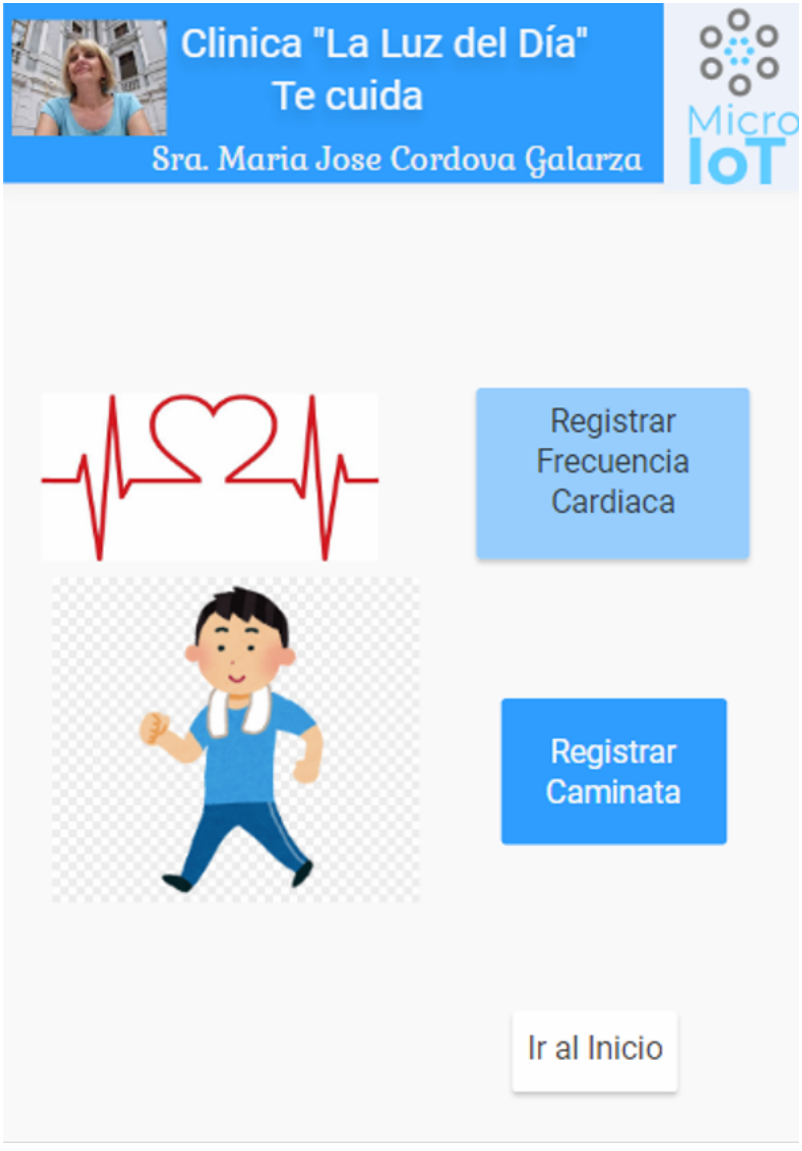


Figura 5.14: Pantalla de inicio de la aplicación móvil Android

Figura 5.15: Maqueta del hogar del paciente con control de la iluminación

5.9. Segunda iteración

Dado que se planificó y acordó con los interesados, que el sistema de software de IoT para AAL, considera dos entregas de software de las cuales la última todavía está pendiente. Por ello, es necesario realizar una segunda iteración de la metodología MicroIoT, cómo se presenta en las siguientes secciones. Como en la entrega anterior los interesados no presentaron nuevo requerimientos para este iteración únicamente se abordará lo que comprende la segunda entrega.

5.10. Análisis de requerimientos de la segunda entrega

Como se especificó en la sección 5.2 el cliente mostró conformidad con la solución presentada en la entrega anterior, además de ello; no solicitó nuevo requerimientos para esta entrega es por ello que no es necesario realizar esta actividad y se puede avanzar a la siguiente.

5.11. Diseño de la entrega dirigida por el dominio para la segunda entrega

Debido a que en el paso anterior no se identificaron nuevos requerimientos se considera únicamente a la segunda entrega para realizar el diseño de la entrega. a continuación se Ejecutan las tareas comprendidas en esta actividad.

5.11.1. Análisis del dominio

En esta tarea MicroIoT recomienda identificar lo siguiente:

A. Identificación del problema

Implementar un mecanismo que ofrezca al usuario una mejor experiencia al controlar la iluminación de su hogar, además ir enviar una alerta al médico en caso de presentarse presentarse alguna anomalía en los signos vitales del paciente.

B. Solución propuesta

La solución planteada por el arquitecto de software misma que fue aprobada por el interesado consiste en lo siguiente:

- Desarrollar una aplicación de escritorio que permita controlar la iluminación del hogar a través de señales gestuales.
- Integrar la aplicación propuesta como un elemento más en la arquitectura desarrollada en la entrega anterior.
- Identificar un mecanismo que permita la activación de una alarma utilizando el controlador existente.
- Agregar el estado de la alarma a la página web existente.
- Permitir la desactivación de la alarma a través de un pulsante

Luego de analizar las partes que componen la solución propuesta, el arquitecto de software determinó que Las nuevas funcionalidades corresponden mecanismos de interacción con los microservicios existentes por lo que no es necesario realizar las fases “Delimitación de contextos”, “Definición de entidades, agregados y servicios” e “Identificación y validación de microservicios”.

C. Requerimientos generales de AAL

Se enfatizó en identificar los requerimientos que se muestran en la Tabla 5.14:

<table>
<tr><th colspan="2">Requerimientos Generales de AAL Identificados</th></tr>
<tr><td>Tipo o tipos de servicio de salud que abarca el sistema.</td><td>Orientación del sistema (usuarios finales)</td></tr>
<tr><td>Se identificaron dos tipos de servicio de salud:
• Asistencia (para iluminación).
• Vigilancia (alerta)</td><td>La nueva entrega está orientada a dos tipos de usuarios finales:
• Paciente
• Profesionales de la salud</td></tr>
<tr><td>Requerimientos de Hardware (Dispositivos de Internet de las Cosas disponibles o necesarios).</td><td>Ambiente en el que opera el sistema</td></tr>
<tr><td>En un acuerdo entre los involucrados, el analista de requerimientos y el arquitecto de software, se definió que se puede adquirir y/o disponer de los siguientes elementos de hardware:
• Cámara para la detección de señales manuales del paciente.
• Un Buzzer que emita un sonido en caso de activarse una alarma.
• Led para señal de alerta.
• Pulsante para desactivar alarma.
• Raspberry pi 3 para controlar la alarma</td><td>• Hogar</td></tr>
<tr><td colspan="2">Plataforma de despliegue de la o las aplicaciones del sistema:
Para la segunda entrega se identificó las siguientes necesidades:
• Incluir en la página web la información de la alarma de emergencia, a esta página web tiene acceso el paciente para consultar y modificar el estado de la iluminación de su hogar.
• Desarrollar una aplicación de escritorio como otra opción para que el usuario pueda modificar el estado de la iluminación de su hogar.</td></tr>
</table>

Tabla 5.14: Características Generales de AAL para la segunda entrega

D. Requerimientos de Calidad

Los requerimientos no funcionales identificados se muestran en la Tabla 5.15.

Aspectos de Calidad de Software Identificados.	
Aspectos de calidad según ISO 25010.	**Estándares de Salud**
Tanto el analista de requerimientos, el arquitecto de software han considerado importantes los siguientes aspectos: • Seguridad. • Usabilidad. • Seguridad. • Usabilidad. • Eficiencia. • Mantenibilidad.	El interesado junto con el arquitecto de software acordaron que no es necesario utilizar estándares de salud debido a que no se requiere para el control de la iluminación.

Tabla 5.15: Aspectos de Calidad de Software para la segunda entrega

5.11.2. Constitución de equipos de trabajo

Debido a que para esta entrega y nos identificó la creación de nuevos microservicios, no es necesario establecer nuevos equipos de trabajo, sin embargo; la nueva aplicación de escritorio a desarrollar para el microservicio hogar deberá ser realizado por el equipo de trabajo encargado de dicho microservicio.

5.11.3. Establecimiento de los modelos de datos

Ya que esta entrega únicamente abarca mecanismos de interacción con el usuario y no modificaciones a nivel de los microservicios, esta actividad no es necesario

5.12. Arquitectura de la solución de la segunda entrega

De igual manera que en la sección 5.4en esta actividad la solución se adapta a una a una Arquitectura de IoT basada en Microservicios para la nueva entrega. Para definir los componentes de la Arquitectura, se tomaron en cuenta los requerimientos establecidos en las tablas 5.14 y 5.15.

5.12.1. Adecuación de la arquitectura de la aplicación

Con las nuevas funcionalidades establecidas para esta entrega la nueva arquitectura de la solución se presenta en la Figura 5.16, es decir; tanto la capa de interfaz de usuario y la capa de microservicios contiene los mismos elementos de la entrega anterior.

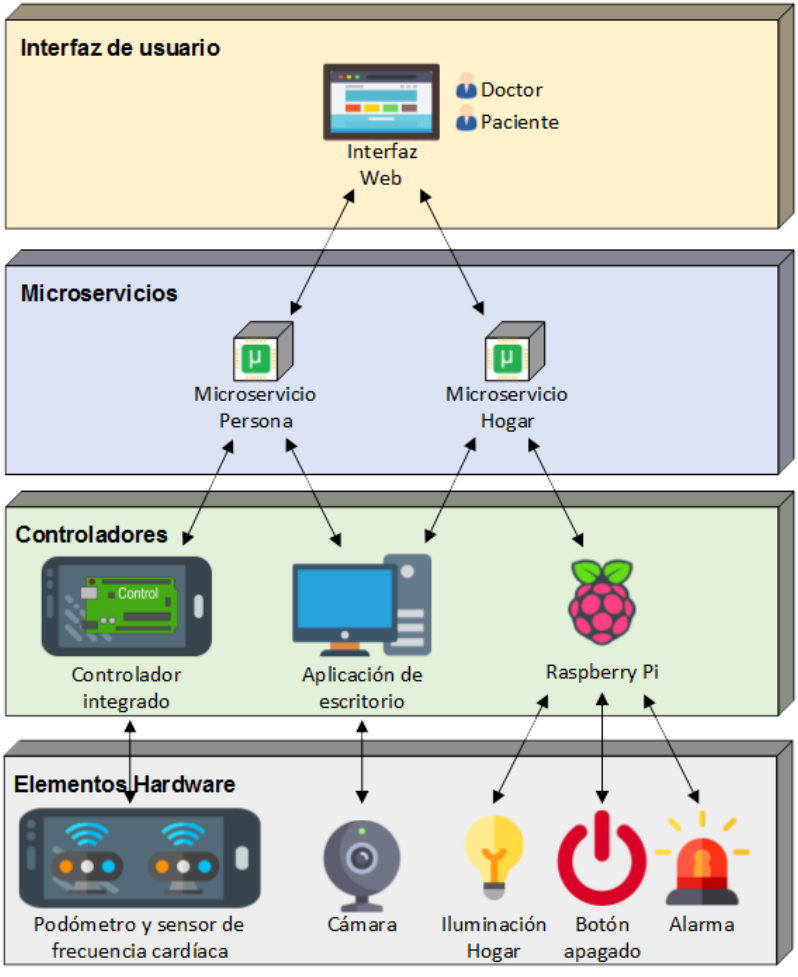


Figura 5.16: Diagrama de la arquitectura de la aplicación para la entrega 2 (Fuente: Elaboración propia).

A. Controladores

Además de los controladores indicados en la entrega anterior, para esta entrega se agrega a la aplicación de escritorio como el controlador que procesará las señales gestuales del paciente para enviarlas al microservicio para invocar al servicio correspondiente de encender o apagar la iluminación.

B. Elementos Hardware

Para esta entrega, se agrega una cámara que permita Durar las imágenes de los gestos del paciente. Además de ello, al Raspberry pi 3 se conecta una alarma y un botón de apagado.

5.12.2. Establecimiento de la arquitectura de gestión de microservicios

Como en esta entrega no se identificó nuevos microservicios, la arquitectura de gestión de microservicios es la misma que en la entrega anterior.

5.13. Validación del diseño y arquitectura de la entrega de la segunda entrega

De igual manera que en la sección 5.5 es necesario validar la entrega de las nuevas funcionalidades del sistema, Por ello el arquitecto de software presenta los nuevos cambios a desarrollar a los interesados, esto se resume en la Tabla 5.16

Criterio de Validación	Propuesta del Arquitecto de Software y Equipo Principal	Retroalimentación de los Interesados
Entendimiento del Problema	Implementar un mecanismo que ofrezca al usuario una mejor experiencia al controlar la iluminación de su hogar, además ir enviar una alerta al médico en caso de presentarse alguna anomalía en los signos vitales del paciente.	✓ Los interesados están de acuerdo con que esa es una vista de alto nivel del problema en cuestión y es lo que necesitan, obviamente se debe entrar en detalle para cada punto en cuestión..

Solución propuesta	La solución consiste en el desarrollo de una aplicación de escritorio la cual permite: • Desarrollar una aplicación de escritorio que permita controlar la iluminación del hogar a través de señales gestuales. • Identificar un mecanismo que permita la activación de una alarma reutilizando el controlador existente. • Agregar el estado de la alarma a la página web existente. • Permitir la desactivación de la alarma a través de un pulsante.	✓ En este caso la respuesta fue satisfactoria, dado que los interesados están de acuerdo con la solución propuesta.
Usuarios finales	• Pacientes • Doctores	✓ Los interesados están de acuerdo.
Vistas o interfaces de usuario	• Integrar en la **página web** la información de la alarma de emergencia.	✓ Los interesados están de acuerdo.
	• **Aplicación de escritorio** a la que accede el paciente para controlar la iluminación del hogar a través de señales gestuales.	✓ Los interesados están de acuerdo.
Ambiente en el que opera el sistema.	• **Hogar:** La iluminación está orientada al hogar del paciente.	✓ Los interesados están de acuerdo.

Aspectos de calidad considerados	Tanto el Analista de Requerimientos, el Arquitecto de Software han considerado importantes los siguientes aspectos: • **Seguridad y Autenticidad.** Se considerará la autenticación de usuarios para la utilización de la aplicación de escritorio. • **Usabilidad.** Se consideran características de usabilidad como la facilidad y eficiencia de uso, además de la naturalidad de la interacción en la aplicación de escritorio. • **Eficiencia.** Se proporciona eficiencia dado que el uso de recursos es adecuado, si bien se incluyen nuevos aspectos de la arquitectura de microservicios estos ayudarán a: i) una mejor distribución de los recursos, basada en la cantidad de peticiones y ii) una escalabilidad independiente para cada funcionalidad del sistema de IoT para AAL requerido. • **Mantenibilidad.** Debido a que es una aplicación que trabaja independiente de los microservicios, esta puede evolucionar para tener nuevas funcionalidades sin afectar a los microservicios.	✓ Los interesados están de acuerdo.
Estándares de datos	• La aplicación de escritorio no posee algún dato relevante que deba ser almacenado, únicamente se enviará señales de control de iluminación.	✓ Los interesados están de acuerdo.

Controladores necesarios	• Controlador raspberry pi 3 Será usado para instalar una alarma (buzzer), una señal de alerta y un botón de apagado. • Un computador sobre el cual se ejecutará la aplicación de escritorio. Por medio de la cámara del computador, de la aplicación de escritorio podrá: i) Interpretar una señal gestual para activar o desactivar la iluminación del hogar.	✓ Los interesados están de acuerdo.
Elementos de Hardware necesarios	• Focos (leds para el prototipo) • Buzzer para señal de alarma. • Cámara para la detección de señales manuales del paciente.	✓ Los interesados están de acuerdo.
Arquitectura del sistema de software de IoT para AAL	Se puede usar el esquema de Arquitectura de Software de la Figura 5.16, que muestra claramente la interacción entre los elementos de hardware y controladores, además de las vistas del sistema de software, así como los usuarios finales o actores que tienen acceso a cada una de las vistas, omitiendo explicaciones técnicas que deben ser transparentes a los interesados, tales como las conexiones de los microservicios con el resto de componentes de la arquitectura.	✓ Los interesados están de acuerdo.

Tabla 5.16: Criterios de validación del diseño y arquitectura de la segunda entrega.

5.14. Desarrollo y Pruebas de la segunda entrega

Como esta entrega no involucra el desarrollo de nuevos microservicios, no se requiere las aplicaciones como los requeridos para el desarrollo y pruebas de la entrega anterior. Para esta entrega se utilizó Python.

5.15. Despliegue de la segunda entrega

Debido a que la aplicación desarrollada en esta entrega es una aplicación de escritorio no es necesario que sea desplegada en la infraestructura por lo que esta actividad no fue realizada.

5.16. Operaciones de la segunda entrega

5.16.1. Monitoreo continuo

Como la entrega contiene únicamente de escritorio no es necesario realizar tareas de monitoreo sobre la misma.

5.16.2. Retroalimentación de los Interesados

Los interesados, han estado presentes durante la mayor parte del proceso aplicativo de MicroIoT, que tiene como objetivo construir un sistema de software IoT para AAL. Durante el ciclo de desarrollo de la MicroIoT no ha surgido nuevos requerimientos ni sugerencias.

A. Validación de la entrega

Luego de completar el ciclo de desarrollo de la metodología se obtienen los productos de software IoT, que cumple con todos los requerimientos especificados para esta entrega. En la validación de la segunda entrega sistema de software de IoT para AAL, realizada en la sección 5.13, se usó una tabla de criterios de validación logrando la aprobación de los interesados. Ahora se valida la segunda entrega codificada, usando los mismos criterios establecidos en la validación del diseño y arquitectura de la entrega, en la Tabla 5.17 se presentan los criterios de validación establecidos para validar los resultados.

Criterio de Validación	Propuesta del Arquitecto de Software y Equipo Principal	Retroalimentación de los Interesados
Entendimiento del Problema	Implementar un mecanismo que ofrezca al usuario una mejor experiencia al controlar la iluminación de su hogar, además ir enviar una alerta al médico en caso de presentarse presentarse alguna anomalía en los signos vitales del paciente.	✓ Los interesados está de acuerdo.
Solución propuesta	La solución consiste en el desarrollo de una aplicación de escritorio la cual permite: • Desarrollar una aplicación de escritorio que Permita controlar la iluminación del hogar a través de señales gestuales. • Identificar un mecanismo que permita la activación de una alarma reutilizando el controlador existente. • Agregar el estado de la alarma a la página web existente. • Permitir la desactivación de la alarma a través de un pulsante	Con respecto a cada punto la retroalimentación de los interesados es la siguiente: ✓ De acuerdo. ✓ De acuerdo. ✓ De acuerdo. ✓ De acuerdo.
Usuarios finales	• Pacientes • Doctores	✓ Los interesados están de acuerdo.
Ambiente en el que opera el sistema	• **Hogar:** La iluminación está orientada al hogar del paciente.	✓ Los interesados están de acuerdo.
Vistas o interfaces de usuario	• **Aplicación web** a la que accede el doctor para ver el estado de sus pacientes, a esta página web también tiene acceso el paciente para consultar y modificar el estado de la iluminación de su hogar	✓ Los interesados están de acuerdo.

	• **Aplicación de escritorio** como otra opción para el usuario para consultar y modificar el estado de la iluminación de su hogar	✓ Los interesados están de acuerdo.
Aspectos de calidad considerados	Los aspectos de calidad que posee la entrega son: • **Seguridad y Autenticidad.** Se considerará la autenticación de usuarios para la utilización de la aplicación de escritorio. • **Usabilidad.** Se consideran características de usabilidad como la facilidad y eficiencia de uso, además de la naturalidad de la interacción en la aplicación de escritorio. • **Eficiencia.** Se proporciona eficiencia dado que el uso de recursos es adecuado, si bien se incluyen nuevos aspectos de la arquitectura de microservicios estos ayudarán a: i) una mejor distribución de los recursos, basada en la cantidad de peticiones y ii) una escalabilidad independiente para cada funcionalidad del sistema de IoT para AAL requerido. • **Mantenibilidad.** Debido a que es una aplicación que trabaja independiente de los microservicios, esta puede evolucionar para tener nuevas funcionalidades sin afectar a los microservicios.	✓ Los interesados están de acuerdo.

Estándares de datos	• Debido a que es una aplicación que trabaja independiente de los microservicios, esta puede evolucionar para tener nuevas funcionalidades sin afectar a los microservicios.	✓ Los interesados están de acuerdo.
Controladores necesarios	• Controlador raspberry pi 3: Usado para instalar una alarma (buzzer), una señal de alerta y un botón de apagado. • Computador: Utilizado para procesar las señales gestuales captadas por la cámara.	✓ Los interesados están de acuerdo.
Elementos de Hardware necesarios	• Focos (leds para el prototipo) • Buzzer como señal de alarma. • Cámara para la detección de señales manuales del paciente.	✓ Los interesados están de acuerdo.
Arquitectura del sistema de software de IoT para AAL	Se puede usar el esquema de Arquitectura de Software de la Figura 5.15, que muestra claramente la interacción entre los elementos de hardware y controladores, además de las vistas del sistema de software, así como los usuarios finales o actores que tienen acceso a cada una de las vistas, omitiendo explicaciones técnicas que deben ser transparentes a los interesados, tales como las conexiones de los microservicios con el resto de componentes de la arquitectura.	✓ Los interesados están de acuerdo.

Tabla 5.17: Criterios de validación del diseño y arquitectura de la entrega.

En la Figura 5.17, se presenta la interfaz de la aplicación de escritorio que permite el control de las luces a través de gestos.

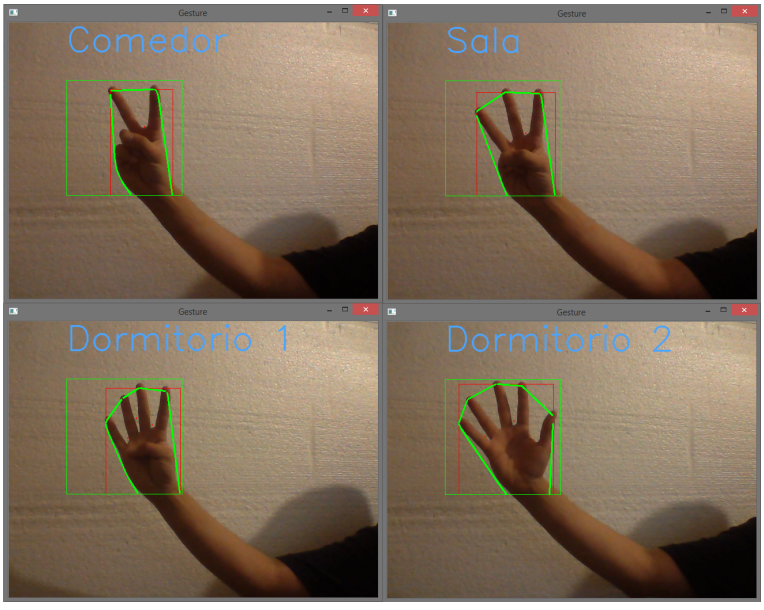


Figura 5.17: Señales gestuales de la aplicación de escritorio para control de Iluminación.

5.17. Conclusiones

En la instanciación de MicroIoT, la metodología propuesta en el Capítulo 4, se evidenció la utilidad de la metodología, las actividades propuestas, se ejecutan correctamente, en un proceso real de diseño, desarrollo y mantenimiento de un software de IoT para AAL, cada actividad abarca los puntos necesarios y suficientes para la creación de una solución de software IoT.

Como parte del proceso de instanciación de MicroIoT se identificaron algunas ventajas de la aplicación de la metodología, entre ellas; la identificación de microservicios fue un proceso intuitivo, el análisis de dominio sirvió de base para la codificación de los microservicios, la arquitectura del sistema se adapta perfectamente a las necesidades de un sistema de software IoT, identificando todos los componentes necesarios y la comunicación entre estos componentes. Los resultados de la aplicación de MicroIoT, para la creación de un sistema de IoT para AAL, basado en un escenario de la vida real, fueron adecuados, dado que el sistema consta de aplicaciones independientes con poca comunicación entre sí, con características de interoperabilidad y mantenibilidad.

Edwin Cabrera Alvarado
Paola Cárdenas Cárdenas

Capítulo 6

Evaluación empírica mediante un caso de uso con cuasi-experimentos

Este capítulo describe la planificación y desarrollo de un cuasi-experimento cuyo objetivo es evaluar las percepciones del usuario frente a la utilidad, facilidad de uso e intención de uso futuro de la metodología propuesta en este trabajo de titulación, descrita en el Capítulo 4, teniendo en cuenta la facilidad generar la arquitectura de aplicación de microservicios para soluciones IoT en ambientes de vida asistidos. En la sección 6.1 se presenta una introducción a la evaluación empírica de la solución propuesta, en la sección 6.2 se muestran los estudios empíricos empleados en el dominio de este trabajo. En la sección 6.3 se muestran los modelos teóricos de evaluación en Ingeniería del Software. La sección 6.4 presenta la adaptación del modelo de evaluación a ser utilizado para evaluar la metodología. La sección 6.5 muestra los cuasi-experimentos desarrollados sobre la solución propuesta. La sección 6.6 presenta las amenazas a la validez y finalmente sección 6.7 presenta las conclusiones de este capítulo.

6.1. Introcucción

Una de las cualidades más destacables en una metodología ágil es su sencillez, tanto en su aprendizaje como en su aplicación, permitiendo a quienes la apliquen obtener un producto de software confiable, de calidad y con un buen diseño frente a los cambios continuos. Por ello es importante evaluar la meto-

dología propuesta en el capítulo 4, para determinar si la metodología cumple las expectativas de los usuarios.

En la Ingeniería de software, los estudios empíricos, se usan para evaluar los procesos, métodos, herramientas o cualquier otro tipo de tecnología creada para el desarrollo, mantenimiento o aseguramiento de la calidad del software (Basili et al., 1986). Un estudio empírico consiste en un acto u operación que permite descubrir algo que no se conoce o poner a prueba una hipótesis (Basili, 1993).

Tomando en cuenta el alto impacto que las percepciones de los usuarios tienen en la selección y adopción de una nueva solución, la aplicación del *Method Evaluation Model* (MEM) (Moody, 2003) puede ser de utilidad a la hora de evaluar la eficacia percibida de MicroIoT. El MEM fue originalmente propuesto como un medio para evaluar métodos de diseño de Sistemas de Información; sin embargo, éste ha sido exitosamente aplicado en la evaluación de otro tipo de métodos (Abrahão et al., 2011). Este modelo ha sido elegido debido a que integra variables del rendimiento actual de los usuarios y de su percepción como mecanismo para predecir la intención de uso y posible adopción en práctica de un método. MEM extiende el *Technology Acceptance Model* (TAM) (Davis, 1985), el cual ha sido validado empíricamente en numerosos estudios que demuestran su utilidad para analizar la facilidad de uso percibida, utilidad percibida e intención de uso de los participantes aplicando un método para predecir su posible aceptación. En este capítulo se presenta la adaptación del MEM para el diseño y ejecución de cuasi-experimentos que proporcionan evidencias sobre la utilidad de MicroIoT.

6.2. Estudios empíricos para metodologías de software existentes

Como se ha visto, el desarrollo de aplicaciones basado en microservicios rompe el esquema de desarrollo de software tradicional; y aún más, el esquema organizacional, siendo una arquitectura que cada vez tiene mayor aceptación. Por ello, es necesario garantizar la eficacia y utilidad de la metodología que se propone. A continuación, se presentan estudios que evalúan diferentes aspectos relacionados a metodologías y a microservicios.

Por un lado, estudios existen estudios como Vavpotič et al. (2004) o Tinoco Gómez et al. (2010), definen técnicas para comparación y selección de metodologías dependiendo del tipo de aplicación a desarrollar, Molina (2014) hace un estudio de las metodologías ágiles más conocidas; sin embargo, en la revisión de la literatura no se encontró una metodología de desarrollo de software centrada en la Arquitectura de Microservicios, menos aún, en un entorno

de Internet de las Cosas para Ambientes de Vida Asistidos. Por ello MicroIoT no puede someterse a un modelo de comparación de metodologías.

Acevedo et al. (2017) propone una metodología para transformar una aplicación monolítica en una arquitectura de microservicios, sin embargo; el estudio no realiza una validación empírica de la metodología.

Villamizar et al. (2015) analiza y prueba el patrón de Arquitectura de Microservicios, presentando un caso de estudio en el cual una aplicación empresarial es desarrollada y desplegada en la nube utilizando una arquitectura monolítica y una arquitectura de microservicios, analiza y compara el rendimiento de ambas aplicaciones y describe los beneficios y retos que las empresas existentes deben enfrentar para implementar arquitecturas basadas en microservicios en sus aplicaciones.

Los estudios descritos anteriormente demuestran limitaciones en cuanto a evaluaciones empíricas de metodologías y microservicios, tales como i) No existe una metodología existente para la creación de microservicios a partir de un conjunto de requerimientos. ii) Las metodologías propuestas realizan una instanciación de la misma pero no se someten a una evaluación empírica.

6.3. Modelos teóricos de evaluación en Ingeniería de Software

El *Technology Acceptance Model* (TAM) propuesto por Davis (1985) es un modelo teórico aplicado para comprender y evaluar la aceptación de los usuarios hacia nuevas tecnologías, basándose en la teoría del comportamiento planeado y la acción razonada. Por otra parte, el *Method Evaluation Model* (MEM) propuesto por Moody (2003) es un modelo que, además de realizar evaluar la aceptación de tecnologías empleando TAM, en base al pragmatismo metodológico permite determinar el nivel de adopción de un sistema de información.

6.3.1. *Technological acceptance model* (TAM)

El TAM constituye una adaptación de la teoría *Theory of Reasoned Action* (TRA) propuesta por Fishbein and Ajzen (1975). TAM usa TRA como una base teórica para especificar los vínculos causales entre dos pensamientos clave: utilidad percibida y facilidad de uso percibida y las actitudes, intenciones y comportamiento de los usuarios al momento de la adopción de una tecnología de computación. TAM es más general que TRA, está diseñado exclusivamente para el comportamiento en el uso del computador, pero éste incorpora hallazgos acumulados desde hace mucho tiempo en investigación de Ingeniería del

Software.

De acuerdo a TRA, el rendimiento de una persona de una conducta específica está determinado por la intención de su comportamiento y la intención de su comportamiento está conjuntamente determinada por la actitud de la persona y las normas subjetivas concernientes al comportamiento en cuestión Davis (1985). El TRA es un modelo general y como tal, este no especifica las creencias operativas para un comportamiento particular. Utilizando TRA se deberían primero identificar las creencias que son salientes para sujetos de acuerdo al comportamiento bajo investigación.

El TAM utiliza dos creencias de adaptadores potenciales, la facilidad de uso percibida y la utilidad percibida de la tecnología como los principales determinantes de las actitudes hacia una nueva tecnología. Estas actitudes influencian las intenciones y de ahí el comportamiento. En este modelo, el uso es modelado como una función directa de la intención. (Ver la Figura 6.1).

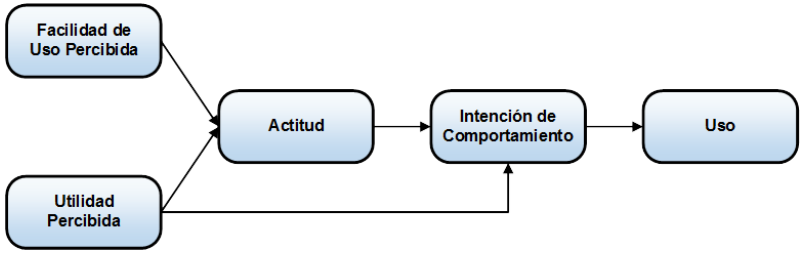


Figura 6.1: Technology Acceptance Model (TAM) simplificado (Davis, 1985)

El significado de cada constructor de TAM es:

- Facilidad de Uso Percibida (PEOU): el grado al cual los usuarios esperan que el sistema objetivo sea libre de esfuerzo.
- Utilidad Percibida (PU): la probabilidad subjetiva del usuario de que utilizando una aplicación específica podría incrementar su rendimiento laboral en un contexto organizacional.
- Actitud (A): el deseo del usuario para usar el sistema. Tanto PU como PEOU predicen la actitud hacia usar el sistema.
- Intención de Comportamiento (IC): la medida de la resistencia a ejecutar un comportamiento específico. A y PU influencian al individuo de IC a usar el sistema
- Uso: el uso actual del sistema. Este es predicho por intenciones del comportamiento.

Además, un gran número de estudios experimentales han validado el TAM usando una variedad de sistemas específicos, tales como la predicción de uso de sitios Web (Fenech, 1998) y el éxito de aplicaciones comerciales existentes (Schubert and Dettling, 2002).

6.3.2. *Method Evaluation Model* (MEM)

La principal contribución del MEM es que este incorpora dos diferentes aspectos del método TAM: la eficiencia actual y el uso actual. La Figura 6.2 ilustra las diferentes partes de este método. Esto significa que la adopción de un método en práctica depende no solamente de si este es efectivo (éxito pragmático) (Abrahão et al., 2011), sino además de si los usuarios de un método lo perciben efectivo (éxito percibido). La Figura 6.2 además muestra los constructores del modelo, así como también las relaciones causales a lo largo de los constructores del mismo.

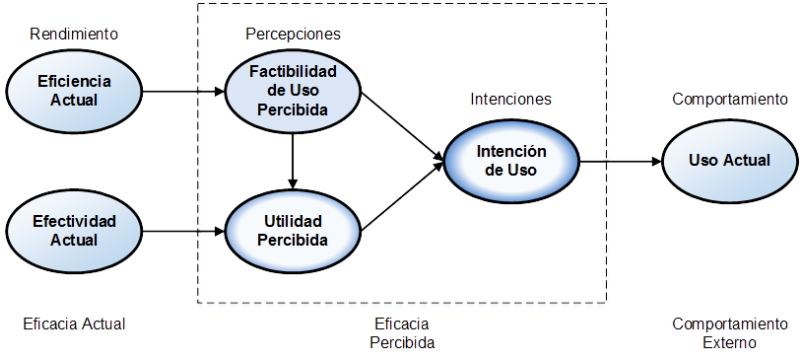


Figura 6.2: Method Evaluation Model MEM (Fuente: (Moody, 2003))

En el MEM, la eficacia es definida como un constructor separado, el cual es diferente de la eficiencia y la efectividad. El constructor de la eficacia es derivado de la noción del éxito pragmático de Rescher (1973), el cual es definido como la eficacia y la efectividad con la cual un método consigue sus objetivos. La evaluación de la eficacia de un método requiere la medición tanto del esfuerzo requerido (eficiencia) y la calidad de los resultados (efectividad). Tomando en cuenta las definiciones dadas por Cedillo (2017) los constructores del MEM están basados en el *Technology Acceptance Model* (TAM) y son los siguientes:

- Eficacia Actual: este constructor tiene dos variables basadas en el rendimiento del usuario:

- Eficiencia Actual: es el esfuerzo requerido para aplicar un método.
- Efectividad Actual: es el grado en el cual un método consigue sus objetivos. Este constructor está relacionado a la calidad de los artefactos obtenidos al momento de aplicar el método. Según Rescher (1973), todos los métodos intentan conseguir ciertos objetivos. Rescher define un método como "una colección de reglas y procedimientos designados para asistir a la gente al momento de ejecutar una acción particular". Diferentes tipos de métodos son definidos para diferentes objetivos. Esto significa que variables específicas dependientes se pueden necesitar para definir cada clase de métodos para medir el rendimiento con respecto a sus objetivos específicos.

- Eficacia Percibida, la cual tiene dos variables basadas en la percepción
- Facilidad de Uso Percibida: es el grado en el cual una persona cree que usando un método en particular puede estar libre de esfuerzo. La facilidad de uso percibida representa un juicio perceptivo del esfuerzo requerido para aprender y usar el método en cuestión.
- Utilidad Percibida: es el grado en el cual una persona cree que usando un método particular podría mejorar su rendimiento en el trabajo. Esta variable representa un juicio perceptual de la efectividad del método. Existe una relación causal en el modelo el cual indica que la utilidad percibida puede estar determinada por la facilidad de uso percibida.
- Intención de Uso: es el modo en el que una persona intenta usar un método particular. Esta variable representa un juicio perceptual de la eficacia y efectividad de coste del método. Esta variable es usada para medir la probabilidad del método para ser adoptado en la práctica. Las relaciones causales sugeridas que perciben la facilidad de uso y la utilidad percibida directamente afectan la intención de usar el método.
- Uso Actual, el cual representa una variable basada en el comportamiento, definida como la intención de utilizar un método en la práctica. De acuerdo a la relación causal hipotética, el uso actual debe estar determinado por la intención del uso.

6.4. Adaptando el MEM para su uso en metodologías de desarrollo de software

Para adaptar el MEM primero se requiere definir los aspectos específicos de la metodología a evaluar, en base a ello; los constructores generales del MEM

pueden ser instanciados en variables dependientes concretas basadas en estos objetivos.

La evaluación empírica se centra en determinar la eficacia y eficiencia del usuario al realizar crear arquitectura de un sistema basado en microservicios aplicando a partir de una lista de requerimientos modelo de dominio establecido de un escenario planteado aplicando la metodología propuesta, es decir; esta evaluación se centra evaluar la eficacia y eficiencia de realizar las actividades "Diseño de entrega guiada por dominio" y "Arquitectura de la solución", las fases de las actividades anteriormente descritas se presentan en la Figura 6.3.

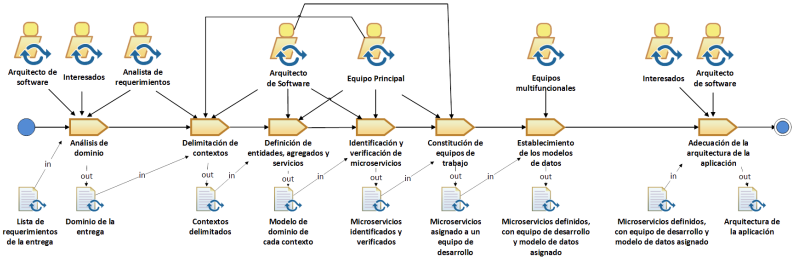


Figura 6.3: Pasos comprendidos en la evaluación empírica (Fuente: Elaboración propia).

La evaluación de la eficacia de la metodología involucra la medición del esfuerzo requerido para crear una arquitectura basada en microservicios usando la metodología en un escenario específico, y la calidad de los resultados obtenida.

El esfuerzo requerido para entender y/o aplicar el método (eficiencia actual) puede ser medido utilizando algunas medidas, tales como el tiempo o el esfuerzo cognitivo. La calidad del resultado del método (actual efectividad) puede ser medida evaluando los resultados de la creación de la arquitectura de la aplicación utilizando los componentes de la arquitectura, además de los resultados de cada actividad que comprende la evaluación

Las siguientes variables de rendimiento son así utilizadas para medir la eficiencia y efectividad de los usuarios con respecto a la configuración de los RNF a ser monitorizados:

- Efectividad actual: A cada paso de la evaluación se asignará una calificación en base al grado de completitud, es decir el grado en que cada tarea haya sido desarrollada correctamente. Esto se calcula mediante la

siguiente fórmula:

$$EfectividadTarea = \frac{\#elementos\_correctamente\_identificados}{\#elementos\_totales\_de\_la\_tarea}$$

La efectividad total de la evaluación será igual a la sumatoria de la efectividad de las tareas.

- Eficiencia actual: La eficiencia del experimento se mide a través del tiempo empleado en cada tarea, la eficiencia del experimento se obtiene aplicando la siguiente ecuación:

$$\sum_{i=1}^{n} tiempo\_para\_desarrollar\_tarea_i$$

Con el objetivo de medir las variables basadas en percepción, se ha adaptado un instrumento de medición utilizado en el MEM. La Figura 6.2 muestra como se ha sido operacionalizado el MEM para la creación de sistemas haciendo uso de la metodología propuesta. Para medir la facilidad de uso percibida, utilidad percibida e intención de uso se ha definido conjuntos de preguntas basados en los ítems mostrados en la Tabla 6.1. La Figura 6.4 muestra la distribución de las preguntas que medirán las percepciones de los usuarios, en donde la facilidad de uso percibida se la representa por PEOU, la utilidad percibida por PU y la intención de uso futura por ITU.

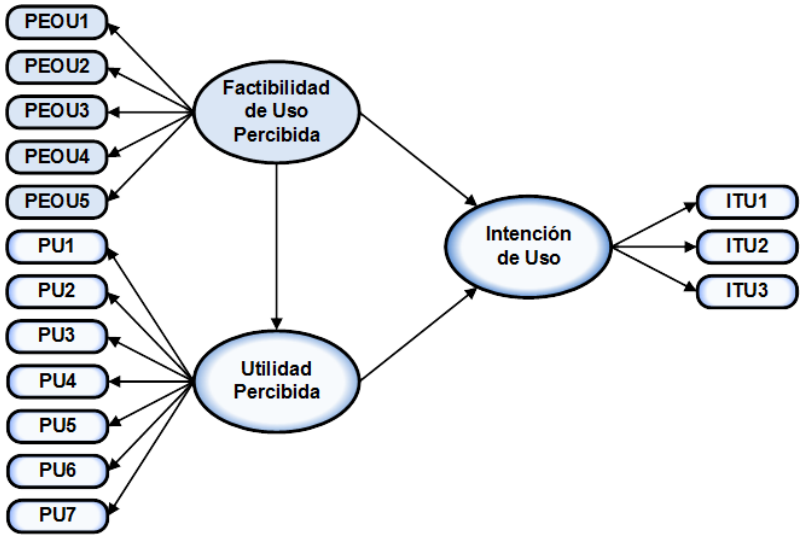


Figura 6.4: Distribución de preguntas del cuestionario que medirá las percepciones de usuario (Fuente: Elaboración propia).

Para determinar si la metodología es aceptada, es necesario probar las hipótesis presentadas en la Figura 6.5 y sus relaciones causales.

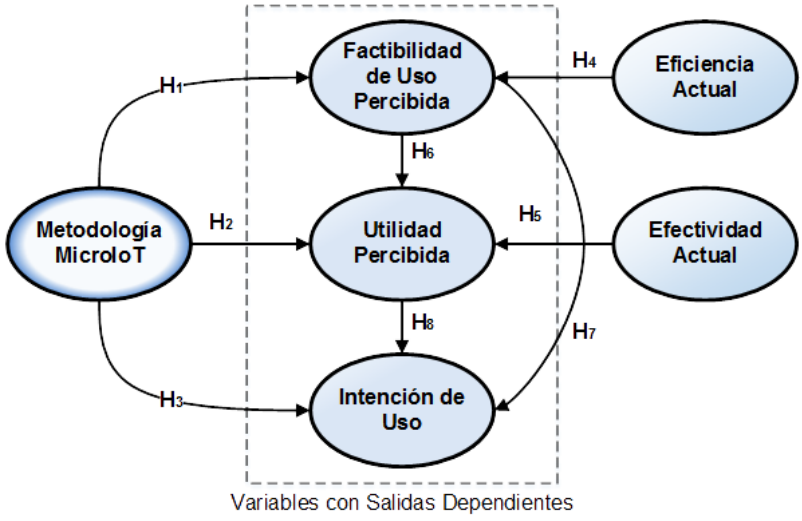


Figura 6.5: Modelo teórico para la evaluación de la metodología Fuente(Cedillo (2014)).

- $H1_0$: La metodología se percibe como difícil de usar, $H1_0$=$\neg H1_1$
- $H2_0$: La metodología no se percibe como útil, $H2_0$ =$\neg H2_1$
- $H3_0$: No existe intención de utilizar la metodología en el futuro $H3_0$=$\neg H3_1$

Además, se plantean hipótesis que muestran una relación directa entre el uso de la metodología y el rendimiento, percepciones e intenciones de los usuarios.

- $H4_0$: La facilidad de uso percibida no puede verse determinada por la eficiencia, $H4_0$=$\neg H4_1$ ya que la eficiencia se mide en base a la eficiencia actual y la facilidad de uso percibida representa una medida en base de la percepción.
- $H5_0$: La percepción de la utilidad no está determinada por la efectividad. $H5_0$=$\neg H5_1$ ya que la efectividad se mide en base al rendimiento mientras que la utilidad percibida se obtiene en base a la percepción de la efectividad.
- $H6_0$: La utilidad percibida no es determinada por la facilidad de uso percibida, $H6_0$=$\neg H6_1$ Esta hipótesis es tomada desde el TAM, en el cual la facilidad de uso percibida se encuentra que no tiene una influencia directa sobre la utilidad percibida.

- $H7_0$: La intención de uso no es determinada por la facilidad de uso percibida, $H7_0$=¬$H7_1$ Esta hipótesis es tomada desde el TAM en el cual se encuentra que la facilidad de uso percibida tiene influencia sobre la intención de uso.

- $H8_0$: La intención de uso no está determinada por la utilidad percibida. $H8_0$=¬8_1 Esta hipótesis es tomada del TAM, en la cual se encontró que la utilidad percibida tiene una influencia directa sobre la intención de uso.

En la Tabla 6.1 se lista las preguntas definidas para medir las variables basadas en la percepción. Estos ítems fueron combinados en un cuestionario con 15 preguntas cerradas valoradas aplicando una escala de Likert del 1 al 5, con el formato de preguntas opuestas. Varios ítems con el mismo grupo de constructores fueron aleatorizados para prevenir errores de respuesta sistemática. La facilidad de uso percibida se mide utilizando cinco ítems del cuestionario. Además, para asegurar el balance de los ítems, ciertas preguntas fueron negadas. El instrumento de medición se encuentra disponible en https://goo.gl/forms/0P7LW6ILVj4FFa2g2.

Pregunta	Declaración positiva
PEOU1	La metodología MicroIoT es sencilla y fácil de seguir.
PEOU2	En general, la metodología es fácil de entender.
PEOU3	Los pasos a seguir para crear la metodología son claros y fáciles de entender.
PEOU4	La metodología es fácil de aprender.
PEOU5	Considero que sería fácil adquirir destrezas en el uso de la metodología.
PU1	Considero que la metodología reduciría el tiempo y el esfuerzo requerido para crear sistemas de este ámbito
PU2	En general, considero que la metodología es útil.
PU3	Considero que la forma de diseño de la aplicación a partir de requerimientos es útil para la creación de sistemas de este ámbito.
PU4	Considero que la forma de establecer la arquitectura del sistema dentro de la metodología MicroIoT es útil para la creación de sistemas de este ámbito.
PU5	Considero que la metodología es lo suficientemente expresiva para definir cómo los componentes interactúan entre sí.
PU6	El uso de esta metodología mejoraría mi rendimiento en la creación de sistemas de este ámbito.
PU7	En general, pienso que esta metodología permite cubrir adecuadamente los requisitos para sistemas de este ámbito.
ITU1	En caso de tener la necesidad de crear sistemas de este ámbito en el futuro, tendría en cuenta esta metodología.
ITU2	De ser necesario, utilizaría esta metodología.
ITU3	Recomendaría el uso de esta metodología.

Tabla 6.1: Cuestionario para medir variables de percepción

Además, se incluyeron dos preguntas abiertas en el cuestionario sobre sugerencias y comentarios, que brindan una retroalimentación para mejorar la arquitectura en los aspectos que allí se mencionen. Las preguntas abiertas se muestran en la Tabla 6.2

Pregunta	Declaración positiva
PA1	¿Tiene alguna sugerencia de cómo hacer que la metodología sea más fácil de usar?
PA2	¿Cuáles son las razones por las que tiene o no la intención de usar esta metodología en un futuro?

Tabla 6.2: Preguntas abiertas del cuestionario.

6.5. Evaluando la utilidad percibida de la metodología MicroIoT en la práctica mediante cuasi-experimentos

En esta sección, se presenta el experimento llevado a cabo para validar empíricamente las actividades "Diseño de entrega guiada por dominio" y "Arquitectura de la solución" de la metodología MicroIoT. En la actualidad no existe, o no se ha encontrado una metodología dirigida a microservicios y que sirva de guía desde la elicitación de requerimientos. Por ello, no se puede evaluar esta metodología con respecto a otra metodología existente. De la metodología MicroIoT, las siguientes actividades no son evaluadas por las siguientes razones:

- La actividad "Análisis de requerimientos" no se realiza para evitar problemas de elicitación de requerimientos.
- La fase "Establecimiento de la arquitectura de gestión de microservicios" perteneciente a la actividad "Arquitectura de la solución" no se evalúa debido a que se requiere personas con conocimiento de microservicios para desarrollar esta actividad.
- La actividad "Validación del diseño de la entrega" es la tarea del evaluador.
- Las actividades comprendidas a partir de la actividad "Desarrollo continuo" hasta la actividad "Monitoreo continuo" no se evalúan ya que son actividades de DevOps y se requiere la instalación de un conjunto de herramientas, así como de personas expertas en este método.

Un cuasi-experimento según (Wohlin et al., 2005) se define como: una investigación empírica en la cual la asignación del tratamiento de los sujetos no se basa en la aleatoriedad, pero emerge desde las características de los sujetos u objetos en sí mismo.

El cuasi-experimento fue diseñado de acuerdo al proceso experimental propuesto por Wohlin et al. (2005). En este caso será utilizado para probar la eficacia percibida de MicroIoT. Para ello, las actividades "Diseño de la aplicación" y "Adecuación de la arquitectura de la aplicación" del cuasi-experimento fueron aplicados para predecir la probabilidad de aceptación de las actividades "Diseño de la entrega dirigida por el dominio" y "Arquitectura de la solución" de nuestra metodología en base a los componentes que se requieren en cada actividad, considerando un estudio basado en las percepciones de los usuarios.

De acuerdo al paradigma *Goal-Question Metric* (GQM) propuesta Basili (1993) la meta de este experimento ha sido definida de la siguiente manera:

- **Evaluar:** Las actividades "Diseño de la entrega dirigida por el dominio" y "Arquitectura de la solución" de la metodología propuesta.
- **Con el propósito de:** evaluar el proceso de diseño aplicaciones basados por microservicios, en base a la metodología propuesta.
- **Desde el punto de vista del:** ingeniero de software que utilizará la metodología.
- **En el contexto de:** un grupo de profesionales y alumnos de años superiores en el área de Ingeniería de Sistemas.

Las preguntas de investigación son:

- RQ1: ¿La metodología propuesta es percibida como fácil de usar y útil ? De ser así, ¿las percepciones de los usuarios son el resultado de su rendimiento cuando utilizan la el método para diseñar sistemas siguiendo una arquitectura basada en microservicios para Internet de las Cosas en Ambientes de Vida Asistidos?
- RQ2: ¿Existe una intención de uso de la metodología de software propuesta en el futuro? De ser así, ¿tales intenciones de uso son el resultado de las percepciones de los participantes?

La primera pregunta de investigación puede ser estudiada mediante las siguientes hipótesis:

- $H1_0$: La metodología es percibida como difícil de usar, $H1_0 = \neg H1_1$
- $H2_0$: La metodología no es percibida como un método útil, $H2_0 = \neg H2_1$
- $H4_0$: La facilidad de uso percibida no puede verse determinada por la eficiencia actual, $H4_0 = \neg H4_1$

- $H5_0$: La percepción de la utilidad no está determinada por la efectividad actual. $H5_0 = \neg H5_1$

La segunda pregunta de instigación puede ser estudiada a través de las siguientes hipótesis:

- $H3_0$: No existe intención de utilizar la metodología en el futuro $H3_0 = \neg H3_1$.
- $H6_0$: La utilidad percibida no es determinada por la facilidad de uso percibida, $H6_0 = \neg 6_1$.
- $H7_0$: La intención de uso no es determinada por la facilidad de uso percibida, $H7_0 = H7_1$.
- $H8_0$: La intención de uso no está determinada por la utilidad percibida. $H8_0 = H8_1$.

6.5.1. Planificación del cuasi-experimento

A. Selección del Contexto

El contexto está determinado por la metodología MicroIoT que será evaluada y la selección de los participantes. El método de monitorización a ser evaluado es el presentado en este trabajo. Aquí, nos centraremos en la actividad de la configuración, la cual es usada para genera el Modelo de Calidad en Tiempo de Ejecución.

Esta actividad es importante debido a la necesidad de facilitar el desarrollo de aplicaciones basadas en microservicios. Los sujetos de ahí, ejecutarán el rol de Arquitecto de Software en las (Ver Figura 6.2) y de Equipos multifuncionales en las siguientes tareas: (i) Análisis de dominio, (ii) delimitación de contexto, (iii) definir entidades, agregados y servicios, (iv) Identificación y verificación de microservicios, (v) constitución de equipos de desarrollo, (vi) establecimiento de los modelos de datos, y (vii) adecuación de la arquitectura de la aplicación.

Finalmente, 30 participantes fueron seleccionados, de los cuales 19 son estudiantes de último año de la carrera de Ingeniería de Sistemas que han tomado cursos de Calidad de Software, Ingeniería de Software y programación Web y tienen un buen conocimiento de modelos de calidad, métricas y conocimientos de programación web. Además, 7 son estudiantes egresados de la carrera de Sistemas y 4 son profesionales en el área de Ingeniería de Sistemas. Tanto egresados como profesionales, actualmente, desarrollan sus labores en análisis, diseño y programación de sistemas. Todos los participantes son, o fueron estudiantes de la Universidad de Cuenca.

B. Tareas experimentales

El cuasi-experimento consistió en elaborar la arquitectura de una aplicación basada en microservicios para un determinado escenario aplicando la metodología propuesta. Para ello, el cuasi-experimento contiene 9 tareas agrupadas en 4 partes. La Figura 6.6 muestra la relación de correspondencia entre las tareas del cuasi-experimento y las actividades de la metodología.

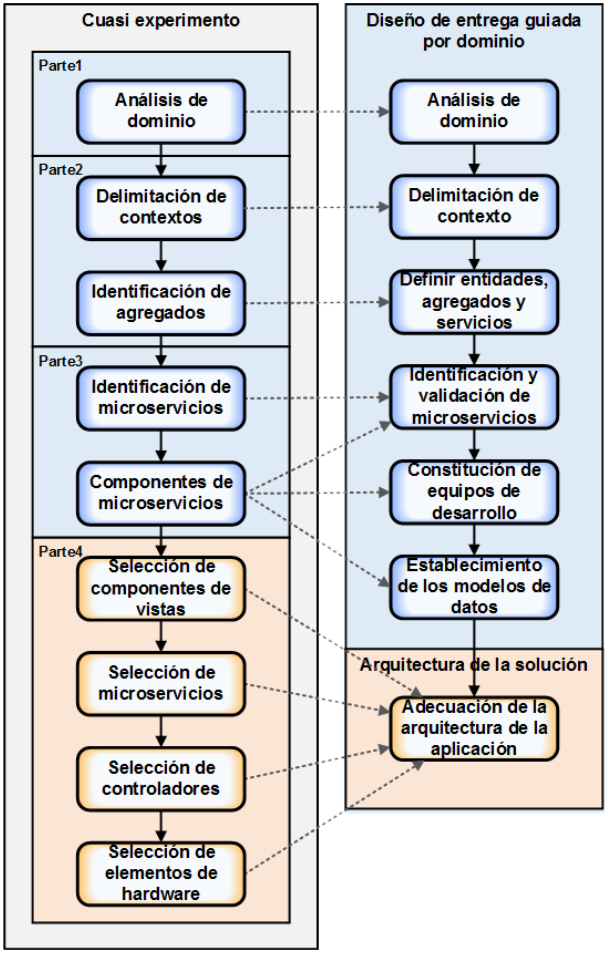


Figura 6.6: Relación entre tareas el cuasi-experimento y actividades de metodología propuesta (Fuente: Elaboración propia).

1. **Tarea 1: Análisis de dominio.** A partir de un problema se establece los diferentes requerimientos para desarrollar la solución. Para evitar problemas de elicitación de requerimientos, esta tarea se entregó resuelto a los participantes por lo que no se considera para medir la facilidad de uso, utilidad percibida y la intención de uso, sin embargo; establece que componentes son requeridos para desarrollar la arquitectura del sistema.

2. **Tarea 2: Delimitación de contextos.** A partir de los requerimientos establecidos, en este paso el problema se descompone en contextos tomando en cuenta que cada contexto cumplirá con un determinado conjunto de requerimientos. Al delimitar un contexto se debe tomar en cuenta que un requerimiento solamente puede ser abarcado por un contexto. En el cuasi-experimento a desarrollar, al participante se le entrega el modelo de dominio de la aplicación siendo una aplicación monolítica. El participante deberá encerrar las clases que considere correspondientes a un contexto, debe tomarse en cuenta que una clase no puede pertenecer a dos contextos diferentes.

3. **Tarea 3: Identificación de agregados.** Dentro de un contexto delimitado, se definen los modelos de dominio de cada contexto. Como el modelo de dominio, fue planteado en la tarea 2 en esta tarea el participante deberá identificar al objeto que se considere el más importante (aquel elemento que más se relacione con las demás clases) dentro de cada contexto, y se lo marcará con la etiqueta *<Agregado>*.

4. **Tarea 4: Identificación de microservicios.** Al tener identificado los contextos, en esta tarea el participante realiza una representación del microservicio mediante un cubo. Es importante que cada contexta posea su microservicio correspondiente.

5. **Tarea 5: Componentes de microservicios.** En esta tarea el participante debe indicar los elementos que tendrá cada microservicio dependiendo de los requerimientos. Los componentes mínimos que debe tener son: *front-end*, base de datos, un equipo de trabajo designado.

6. **Tarea 6: Vistas del sistema y usuarios finales.** Esta tarea inicia la evaluación de la actividad "Arquitectura de la solución" correspondiente a la metodología propuesta, en esta tarea el participante debe indicar el tipo de interfaz de usuario a implementar y el tipo de que utilizará la interfaz

7. **Tarea 7: Microservicios y conexiones.** En esta tarea el participante deben indicar los microservicios identificados en la tarea 4 además de ello,

el microservicio debe vincularse con una determinada vista dependiendo del problema a resolver.

8. **Tarea 8: Controladores y conexiones.** Los controladores forman parte del ambiente IoT para vincular los elementos de hardware es por ello que, en esta actividad el participante debe establecer los tipos de controladores requeridos (se especifican en la tarea 1) y vincularlos con el microservicio que vaya a implementar la funcionalidad requerida.

9. **Tarea 9: Elementos de hardware y conexiones.** Los elementos de hardware constituyen cualquier sensor o actuador que se requiera en el sistema, por ello; el participante deberá indicar los elementos de hardware requeridos y vincularlos con el controlador respectivo (el controlador se especifica en la tarea 1).

C. Variables

En la Tabla 6.3 se presentan las variables dependientes de interés basadas en la percepción, de acuerdo al MEM, las cuales fueron usadas para evaluar la metodología propuesta en la práctica.

Variable	Descripción
Facilidad de Uso Percibida (PEOU)	El grado en el cual los participantes creen que al aprender y usar MicroIoT estarán libres de esfuerzo.
Utilidad Percibida (PU)	El grado en el cual los participantes creen que usando MicroIoT se incrementará su rendimiento
Intención de uso(ITU)	El grado en el cual los participantes piensan usar MicroIoT en caso de necesitar una metodología para crear aplicaciones basadas en microservicios, para soluciones de IoT dentro de AAL en el futuro. Esto representa un juicio de la eficacia del método y puede ser utilizado para predecir la aceptación del método en la práctica.

Tabla 6.3: Variables dependientes basadas en la percepción.

Estas variables son medidas usando un cuestionario con una escala de Likert con un con un conjunto de 15 preguntas cerradas (ej. 5 para facilidad de uso

percibida, 6 para utilidad percibida y 3 para intención de uso futura). Las preguntas cerradas fueron formuladas utilizando una escala de Likert de 5 puntos. El valor agregado para cada variable subjetiva fue calculado como la media aritmética de las respuestas a las preguntas asociadas con cada variable dependiente subjetiva. En la Tabla 6.4 se muestran las variables basadas en el rendimiento de interés y la función de medición usada para determinar sus valores.

Variable	Descripción
Efectividad	$Efectividad = \frac{\sum_{i=1}^{n} Efectividad\_Tarea_i}{n}$
Eficiencia	$Eficiencia = \sum_{i=1}^{n} Tiempo\_empleado\_en\_parte_i$

Tabla 6.4: Variables dependientes basadas en el rendimiento.

D. Material Experimental

El material experimental se compone del conjunto de documentos que son necesarios para realizar las tareas experimentales y el cuestionario para medir la percepción del usuario una vez que se ha realizado el experimento. Este material incluye todo lo referente a las presentaciones con los conceptos, explicación de tareas y ejemplos que son utilizadas para el entrenamiento de los participantes. La documentación utilizada, en su totalidad, ha sido incluida en los anexos de este trabajo y es explicada a continuación:

1. Un folleto que contiene la descripción del proceso de evaluación, empleando el ejemplo resuelto, ilustra la forma de desarrollar las tareas propuestas para para los participantes. Se solicitó a los participantes que escriban la hora exacta de inicio y fin de cada una de las tareas a resolver. (revisar Anexo A.6)

2. Un anexo detallado como soporte, el cual describe brevemente las actividades de la metodología (incluyendo las actividades no cubiertas por el cuasi-experimento). (revisar Anexo A.7)

3. Un ejemplo resuelto el cual que sirve de guía para la resolución del cuasi-experimento. (revisar Anexo A.8)

4. Hoja de resolución para que el participante plasme sus respuestas. (revisar Anexo A.9)

5. Un cuestionario que contiene preguntas cerradas para analizar las variables subjetivas y algunas respuestas abiertas para permitir a los participantes expresar su opinión sobre la metodología. (revisar Anexo A.10)

6. Además de ello, como referencia, en el Aneo A.11 se adjunta la solución al ejercicio propuesto durante el cuaso-experimento.

El tiempo gastado en cada tarea y los datos obtenidos como resultado fueron recolectados utilizando las hojas de resolución descritas en el numeral cuatro. El cuestionario fue recolectado en línea. Todos los documentos fueron creados en español, ya que éste es el idioma nativo de los participantes en todos los experimentos.

6.5.2. Operación y ejecución de los cuasi-experimentos

Se realizaron dos experimentos individuales, para cada uno de ellos se efectuó una sesión de entrenamiento de 20 minutos antes de la sesión experimental con la finalidad de presentar los conceptos relacionados a la metodología de software propuesta, explicar el material experimental y desarrollar un ejemplo demostrativo. La Figura 6.7 brinda un resumen de los cuasi-experimentos realizados.

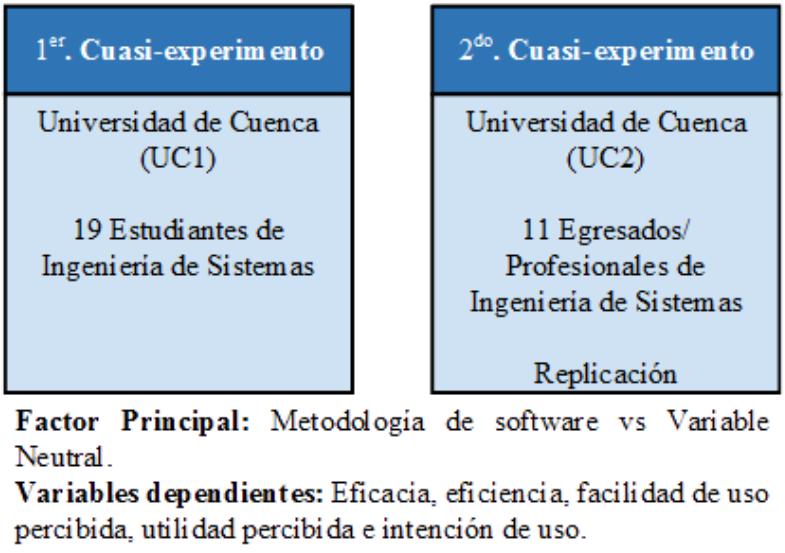


Figura 6.7: Resumen de los cuasi-experimentos realizados.

En el cuasi-experimento original (estudiantes) participaron 19 estudiantes de la carrera de Ingeniería de Sistemas de la Universidad de Cuenca que actualmente cursan el último año de la carrera. El segundo cuasi-experimento corresponde a una réplica del cuasi-experimento original y se realizó con 11 sujetos que actualmente se encuentran en el ámbito profesional y anteriormente pertenecieron a la universidad de Cuenca.

6.5.3. Ejecución y análisis de los requerimientos individuales

Para realizar los análisis y establecer criterios de aceptación o rechazo de hipótesis, se han utilizado los niveles de significancia sugeridos por Moody (2003) los cuales se muestra en la Tabla 6.5.

Valor de Significancia	Rango
No significativo	p >0.1
Baja significancia	p <0.1
Media significativo	p <0.05
Alta significancia	p <0.01
Muy alta significancia	p <000.1

Tabla 6.5: Niveles de significancia (Moody, 2003)

A. Cuasi-experimento UC1 (Estudiantes)

Este cuasi-experimento mencionado fue realizado en la Universidad de Cuenca con un total de 19 estudiantes en una sesión de 60 minutos comprendida por 20 minutos de entrenamiento y 40 minutos para la ejecución del cuasi-experimento. Durante la sesión, cualquier inquietud de los participantes podía ser solventada por los evaluadores, sin embargo; no se presentaron inquietudes relevantes a lo largo del cuasi-experimento. Al finalizar el cuasi-experimento se pidió a los participants que llenen el cuestionario post-experimento. Para el análisis de los resultados, se usaron pruebas, estadística descriptiva y *box plots* sobre los datos recolectados. Los resultados fueron obtenidos usando SPSS v23 con un α=0.05.

I. **Análisis de las percepciones de usuario**
La Figura 6.8 muestra los diagramas de caja para cada variable de percepción en donde se observa que la media de cada variable es mayor que 3, que representa al valor neutro de la escala de Likert.

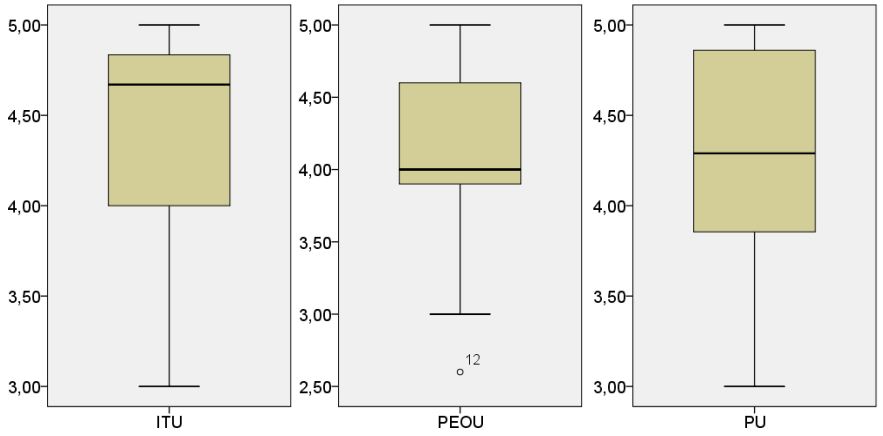


Figura 6.8: Diagramas de caja correspondientes al cuasi-experimento UC1.

Posteriormente se determinó si los datos siguen una distribución normal a través de la prueba de normalidad de Shapiro-Wilk, para luego seleccionar el test adecuado que permita chequear las hipótesis H1, H2 y H3.

Como se aprecia en la última columna de la Tabla 6.6 las variables PEOU y PU siguen una distribución normal (p-value>0.05), por lo que; se aplicó el *t-test one-tailed* con un valor de prueba igual a 3, ya que este valor corresponde al valor neutral de la escala Likert del cuestionario. Por otra parte, como la variable ITU no sigue una distribución normal (p<0.05) se aplicó el test *Wilcoxon one tailed one sample* igualmente con un valor de prueba igual a 3. Con estas pruebas se determina si se aceptan o rechazan las hipótesis nulas $H1_0$, $H2_0$ y $H3_0$.

Var	**Min**	**Max**	**Media**	**Std. Dev**	**Std E.**	**1-T. p-value**	**Shapiro-wilk test p-value**
PEOU	2.60	5	4.1053	0.64428	0.14779	0.000 <0.01	0.131>0.05
PU	3	5	4.2337	0.58483	0.13417	0.000 <0.01	0.259 >0.05
ITU	3	5	4.4042	0.55197	0.12893	0.000 <0.01	0.033 <0.05

Tabla 6.6: Resultados de análisis de percepciones del usuario (UC1).

De los resultados presentados en la Tabla 6.6, en la columna "1-T. p-value" se presenta los resultados de las pruebas *t-test one-tailed* y el test de *Wilcoxon*, estos valores indican el nivel de significancia que indica, el

cual en base a la Tabla 6.5 indica un muy alto grado de significancia. Esto permite rechazar las hipótesis nulas $H1_0$, $H2_0$ y $H3_0$, lo que significa que MicroIoT es percibida como fácil de usar, útil y que los participantes consideran usarla en el futuro cuando tengan que diseñar sistemas basados en microservicios para ambientes de IoT en AAL.

II. **Análisis de rendimiento del usuario**
Se midió la efectividad y la eficiencia de los participantes cuando utilizan MicroIoT en la práctica. La Tabla 6.7 muestra los valores de estadística descriptiva basada en el rendimiento. Los participantes tuvieron un promedio de 79.74 % de efectividad al completar el ejercicio.

La eficiencia, ha sido calculada como el esfuerzo requerido en minutos para desarrollar el método (Moody, 2003). Los participantes completaron el experimento entre 6 a 28 minutos. Esto depende de varios factores como la experiencia de los participantes en el ámbito de microservicios y sus conceptos relacionados.

Variable	Min	Max	Media	Desviación Std
Efectividad	0.60	0.96	0.7974	0.10434
Eficiencia	6.00	28.00	14.3158	6.72518

Tabla 6.7: Estadística descriptiva para variables usadas en el Rendimiento del Usuario (UC1)

III. **Análisis de relaciones causales** El objetivo de esta sección es validar la parte estructural del MEM en términos de las relaciones causales entre sus constructores, con la excepción del Uso Actual. Para esto, se ha utilizado análisis de regresión para evaluar la operacionalización del MEM realizada, ya que las hipótesis a ser probadas son relaciones causales entre variables continuas. Para realizar este análisis, se ha utilizado los niveles de significancia sugeridos por Moody (2003), los cuales se ilustraron en la Tabla 6.5.

a. **Eficiencia VS Facilidad de uso Percibida**
La hipótesis H4 se ha definido con el objetivo de comprobar si las percepciones de Facilidad de Uso Percibida (PEOU) son determinadas por la Eficiencia de los participantes al usar la metodología. El análisis consiste en la construcción de un modelo de regresión simple en donde la Eficiencia es la variable independiente y PEOU la variable dependiente. La ecuación de regresión resultante del análisis es la siguiente:

$$PEOU = 4{,}109 + (0) * Eficiencia$$

Reg. Elemento	Coef (b)	Std. E	Std. Coef	t	Sig(p)	R	R^2
Constante	4.109	0.366		11.237	0.000		
Eficiencia	0.000	0.023	-0.003	-0.012	0.990	0.003	0.000

Tabla 6.8: Regresión simple entre Eficiencia y Facilidad de uso Percibida (UC1)

Como se muestra en la 6.8, el modelo de regresión se considera no significativo ya que $p > 0{,}1$. R^2 muestra que la variable eficiencia influye en un 0 % sobre PEOU, en otras palabras; la eficiencia de los participantes no influencia sobre la facilidad de uso percibida. Este resultado no nos permite rechazar la hipótesis nula $H4_0$ y aceptar la hipótesis alternativa.

b. **Efectividad VS Utilidad Percibida**
La hipótesis H5 ha sido probada para verificar si las percepciones de la Utilidad Percibida (PU) están determinadas por la Efectividad de los participantes. Similarmente, nosotros construimos un modelo de regresión simple en el cual la Efectividad fue usada como la variable independiente y PU es la variable dependiente. La ecuación de regresión aplicada es la siguiente:

$$PU = 4{,}445 + (-0{,}264) * Efectividad$$

Reg. Elemento	Coef (b)	Std. E	Std. Coef	t	Sig(p)	R	R^2
Constante	4.445	1.091		4.072	0.01		
Efectividad	-0.264	1.358	-0.047	-0.195	0.848	0.047	0.02

Tabla 6.9: Regresión simple entre Efectividad y Utilidad Percibida (UC1)

Como se muestra en la Tabla 6.9, el modelo de regresión se considera no significativo ya que $p > 0{,}1$. R^2 muestra que la variable efectividad influye en un 2 % sobre PU, en otras palabras; la efectividad de los participantes al realizar el cuasi-experimento no influye sobre la utilidad que perciben. Este resultado no nos permite rechazar la hipótesis nula $H5_0$ y aceptar la hipótesis alternativa.

c. **Facilidad de Uso Percibida VS Utilidad Percibida**
La hipótesis H6 ha sido probada para verificar si las percepciones de

la Utilidad Percibida (PU) están determinadas por la Facilidad de Uso Percibida (PEOU). Similarmente, nosotros construimos un modelo de regresión simple en el cual la PEOU fue usada como la variable independiente mientras que PU es la variable dependiente. La ecuación de regresión aplicada es la siguiente:

$$PU = 1{,}391 + (0{,}693) * PEOU$$

Reg. Elemento	**Coef (b)**	**Std. E**	**Std. Coef**	**t**	**Sig(p)**	**R**	R^2
Constante	1.391	0.291		2.352	0.31		
PEOU	0.693	0.142	0.763	4.864	0.000	0.763	0.582

Tabla 6.10: Regresión simple entre Facilidad de Uso Percibida y Utilidad Percibida (UC1).

Como se muestra en la Tabla 6.10, el modelo de regresión se considera que tiene muy alta significancia ya que $p < 0{,}01$. R^2 muestra que la variable efectividad influye en un 58.2 % sobre PU, en otras palabras; La utilidad percibida (PU) está determinada por la facilidad de uso percibida (PEOU) por los participantes la facilidad de uso. Este resultado nos permite rechazar la hipótesis nula $H6_0$ y aceptar su hipótesis alternativa.

d. **Intención de uso VS Facilidad de uso Percibida**
La hipótesis H7 ha sido probada para verificar si las percepciones de la Intención de Uso (ITU) están determinadas por la Facilidad de Uso Percibida (PEOU). Similarmente, nosotros construimos un modelo de regresión simple en el cual la PEOU fue usada como la variable independiente mientras que ITU es la variable dependiente. La ecuación de regresión aplicada es la siguiente:

$$ITU = 2{,}718 + (0{,}411) * PEOU$$

Reg. Elemento	**Coef (b)**	**Std. E**	**Std. Coef**	**t**	**Sig(p)**	**R**	R^2
Constante	2.718	0.775		3.506	0.003		
PEOU	0.411	0.187	0.471	2.200	0.42	0.471	0.222

Tabla 6.11: Regresión simple entre Intención de Uso y Facilidad de Uso Percibida (UC1)

Como se muestra en la Tabla 6.11, el modelo de regresión se considera que no tiene significancia ya que $p > 0{,}1$. R^2 muestra que la PEOU influye en un 22.2 % sobre ITU, en otras palabras; la facilidad de uso percibida de los participantes tiene influencia sobre sobre la intención de uso. Este resultado no nos permite rechazar la hipótesis nula $H7_0$ y aceptar la hipótesis alternativa.

e. **Utilidad Percibida VS Intención de uso**
La hipótesis H8 ha sido probada para verificar si las percepciones de la Intención de Uso (ITU) están determinadas por la Utilidad Percibida (PU). Similarmente, nosotros construimos un modelo de regresión simple en el cual la PU fue usada como la variable independiente mientras que ITU es la variable dependiente. La ecuación de regresión aplicada es la siguiente:

$$ITU = 1{,}892 + (0{,}593) * PU$$

Reg. Elemento	**Coef (b)**	**Std. E**	**Std. Coef**	**t**	**Sig(p)**	**R**	R^2
Constante	1.892	0.783		2.416	0.27		
PU	0.593	0.183	0.618	3.237	0.05	0.618	0.381

Tabla 6.12: Regresión simple entre Utilidad Percibida e Intención de uso(UC1).

Como se muestra en la Tabla 6.12, el modelo de regresión se considera de significancia media ya que $p = 0{,}05$. R^2 muestra que la PU influye en un 38.1 % sobre ITU, en otras palabras; la utilidad percibida de los participantes tiene influencia sobre sobre la intención de uso. Este resultado nos permite rechazar la hipótesis nula $H8_0$ y aceptar la hipótesis alternativa.

B. Cuasi-experimento UC2 (Egresados y profesionales)

Este cuasi-experimento fue realizado en la Universidad de Cuenca con un total de 11 participantes. El procedimiento fue el mismo que el usado durante la ejecución y análisis del cuasi-experimento original (estudiantes).

De forma similar que el cuasi-experimento UC1 se determinó si los datos siguen una distribución normal a través de la prueba de normalidad de Shapiro-Wilk, para luego seleccionar el test adecuado que permita chequear las hipótesis H1, H2 y H3.

I. **Análisis de las percepciones de usuario**
La Figura 6.9 muestra los diagramas de caja para cada variable de percepción en donde se observa que la media de cada variable es mayor que 3, que representa al valor neutro de la escala de Likert.

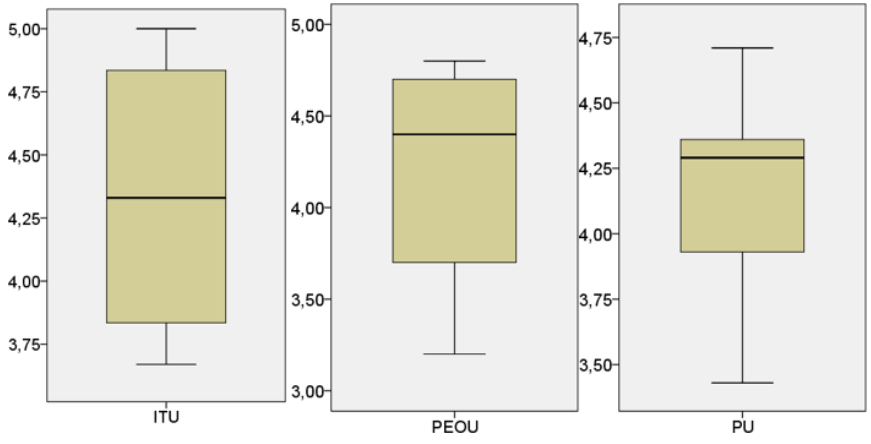


Figura 6.9: Diagramas de caja correspondientes al cuasi-experimento UC2.

Como se aprecia en la última columna de la Tabla 6.13 las variables PEOU, PU e ITU siguen una distribución normal (p-value>0.05), por lo que; se aplicó el t-test one-tailed con un valor de prueba igual a 3, ya que este valor corresponde al valor neutral de la escala Likert del cuestionario. Con estas pruebas se determina si se aceptan o rechazan las hipótesis nulas $H1_0$, $H2_0$ y $H3_0$.

Var	Min	Max	Media	Std. Dev	Std E.	1-T. p-value	Shapiro-wilk test p-value
PEOU	3.20	4.80	41.636	0.61200	0.18453	0.000 <0.001	0.070>0.05
PU	3.43	4.71	4.17	0.36497	0.11004	0.000 <0.001	0.818 >0.05
ITU	3.67	5.00	43.336	0.53646	0.16175	0.000 <0.001	0.078 >0.05

Tabla 6.13: Resultados de análisis de percepciones del usuario (UC2).

De los resultados presentados en la Tabla 6.13, en la columna "1-T. p-value" se presenta los resultados de las prueba *t-test one-tailed* donde

se indica un muy alto nivel de significancia. Esto permite rechazar las hipótesis nulas $H1_0$, $H2_0$ y $H3_0$, lo que significa que MicroIoT es percibida como fácil de usar, útil y que los participantes consideran usarla en el futuro cuando tengan que diseñar sistemas basados en microservicios para ambientes de IoT en AAL.

II. **Análisis del rendimiento del usuario**
Similar al cuasi-experimento UC1, se midió la efectividad y la eficiencia de los egresados e ingenieros cuando usan MicroIoT en la práctica. La Tabla 6.14 muestra los valores de estadística descriptiva basada en el rendimiento. Los participantes tuvieron un promedio de 94.36 % de efectividad al completar el ejercicio.

En cuanto a la eficiencia, los participantes completaron el experimento entre 10 a 31 minutos. Esto depende de varios factores como la experiencia de los participantes en el ámbito de microservicios y sus conceptos relacionados.

Variable	Min	Max	Media	Desviación Std
Efectividad	0.79	1.00	0.9436	0.07775
Eficiencia	10.00	31.00	174.545	711.848

Tabla 6.14: Estadística descriptiva para variables usadas en el Rendimiento del Usuario (UC2)

III. **Análisis de relaciones causales**
El objetivo de esta sección es validar la parte estructural del MEM en términos de las relaciones causales entre sus constructores, con la excepción del Uso Actual. Para esto, se ha utilizado análisis de regresión para evaluar la operacionalización del MEM realizada, ya que las hipótesis a ser probadas son relaciones causales entre variables continuas. Para realizar este análisis, se ha utilizado los niveles de significancia sugeridos por Moody (2003), los cuales se ilustrarón en la Tabla 6.5.

a. **Eficiencia VS Facilidad de uso Percibida**
La hipótesis H4 comprueba si las percepciones de Facilidad de Uso Percibida (PEOU) son determinadas por la Eficiencia de los participantes al usar la metodología. El análisis consiste en la construcción de un modelo de regresión simple en donde la Eficiencia es la variable independiente y PEOU la variable dependiente. La ecuación de regresión resultante del análisis es la siguiente:

$$PEOU = 4{,}288 + (-0{,}007) * Eficiencia$$

Reg. Elemento	Coef (b)	Std. E	Std. Coef	t	Sig(p)	R	R^2
Constante	4.288	0.535		8.018	0.000		
Eficiencia	-0.007	0.029	-0.083	-0.250	0.808	0.083	0.007

Tabla 6.15: Regresión simple entre Eficiencia y Facilidad de uso Percibida (UC2)

Como se muestra en la Tabla 6.15, el modelo de regresión se considera no significativo ya que $p > 0{,}1$. R^2 muestra que la variable eficiencia influye en un 0.7 % sobre PEOU, en otras palabras; la eficiencia de los participantes no influencia sobre la facilidad de uso percibida. Este resultado no nos permite rechazar la hipótesis nula $H4_0$ y aceptar la hipótesis alternativa.

b. **Efectividad VS Utilidad Percibida**
La hipótesis H5 permite verificar si las percepciones de la Utilidad Percibida (PU) están determinadas por la Efectividad de los participantes. Similarmente, nosotros construimos un modelo de regresión simple en el cual la Efectividad fue usada como la variable independiente y PU es la variable dependiente. La ecuación de regresión aplicada es la siguiente:

$$PU = 1{,}359 + (2{,}979) * Efectividad$$

Reg. Elemento	Coef (b)	Std. E	Std. Coef	t	Sig(p)	R	R^2
Constante	1.359	1.144		1.187	0.266		
Efectividad	2.979	1.209	0.635	2.464	0.036	0.635	0.403

Tabla 6.16: Regresión simple entre Efectividad y Utilidad Percibida (UC2)

Como muestra la Tabla 6.16, el modelo de regresión se considera con significancia media ya que $p < 0{,}05$. R^2 muestra que la variable efectividad influye en un 40.3 % sobre PU, en otras palabras; la efectividad de los participantes al realizar el cuasi-experimento no influye sobre la utilidad que perciben. Este resultado nos permite rechazar la hipótesis nula $H5_0$ y aceptar la hipótesis alternativa.

c. **Facilidad de Uso Percibida VS Utilidad Percibida**
La hipótesis H6 permite verificar si las percepciones de la Utilidad

Percibida (PU) están determinadas por la Facilidad de Uso Percibida (PEOU). Similarmente, nosotros construimos un modelo de regresión simple en el cual la PEOU fue usada como la variable independiente mientras que PU es la variable dependiente. La ecuación de regresión aplicada es la siguiente:

$$PU = 3{,}205 + (0{,}232) * PEOU$$

Reg. Elemento	**Coef (b)**	**Std. E**	**Std. Coef**	**t**	**Sig(p)**	**R**	R^2
Constante	3.205	0.770		4.162	0.002		
PEOU	0.232	0.183	0.389	1.265	0.238	0.389	0.151

Tabla 6.17: Regresión simple entre Facilidad de Uso Percibida y Utilidad Percibida (UC2).

Como muestra la Tabla 6.17, el modelo de regresión se considera que no tiene significancia ya que $p > 0{,}1$. R^2 muestra que la variable efectividad influye en un 15.1 % sobre PU, en otras palabras; La utilidad percibida (PU) está determinada por la facilidad de uso percibida (PEOU) por los participantes la facilidad de uso. Este resultado no nos permite rechazar la hipótesis nula $H6_0$ y aceptar su hipótesis alternativa.

d. **Intención de uso VS Facilidad de uso Percibida**
La hipótesis H7 permite verificar si las percepciones de la Intención de Uso (ITU) están determinadas por la Facilidad de Uso Percibida (PEOU). Similarmente, nosotros construimos un modelo de regresión simple en el cual la PEOU fue usada como la variable independiente mientras que ITU es la variable dependiente. La ecuación de regresión aplicada es la siguiente:

$$ITU = 2{,}48 + (0{,}445) * PEOU$$

Reg. Elemento	**Coef (b)**	**Std. E**	**Std. Coef**	**t**	**Sig(p)**	**R**	R^2
Constante	2.480	1.058		2.334	0.044		
PEOU	0.445	0.252	0.508	1.769	0.111	0.508	0.258

Tabla 6.18: Regresión simple entre Intención de Uso y Facilidad de Uso Percibida (UC2)

Como muestra la Tabla 6.18, el modelo de regresión se considera que no tiene significancia ya que baja ya que $p > 0{,}1$. R^2 muestra que la PEOU influye en un 25.0 % sobre ITU, en otras palabras; la facilidad de uso percibida de los participantes tiene influencia sobre sobre la intención de uso. Este resultado no nos permite rechazar la hipótesis nula H70 y aceptar la hipótesis alternativa.

e. **Utilidad Percibida VS Intención de uso**
La hipótesis H8 permite verificar si las percepciones de la Intención de Uso (ITU) están determinadas por la Utilidad Percibida (PU). Similarmente, nosotros construimos un modelo de regresión simple en el cual la PU fue usada como la variable independiente mientras que ITU es la variable dependiente. La ecuación de regresión aplicada es la siguiente:

$$ITU = -1{,}635 + (1{,}409) * PU$$

Reg. Elemento	**Coef (b)**	**Std. E**	**Std. Coef**	**t**	**Sig(p)**	**R**	R^2
Constante	-1.635	1.728		-0.946	0.369		
PU	1.409	0.413	0.751	3.413	0.008	0.751	0.564

Tabla 6.19: Regresión simple entre Utilidad Percibida e Intención de uso(UC2).

Como muestra la Tabla 6.19, el modelo de regresión se considera de significancia alta ya que $p < 0{,}01$. R^2 muestra que la PU influye en un 56.4 % sobre ITU, en otras palabras; la utilidad percibida de los participantes tiene influencia sobre sobre la intención de uso. Este resultado nos permite rechazar la hipótesis nula H80 y aceptar la hipótesis alternativa.

6.5.4. Análisis de los resultados

En esta sección se discuten los resultados de los experimentos en resumen, con el fin de poder contrastar entre ellos y ver posibles semejanzas y diferencias. La Tabla 6.20 presenta cada una de las variables con los valores obtenidos, su media y desviación estándar.

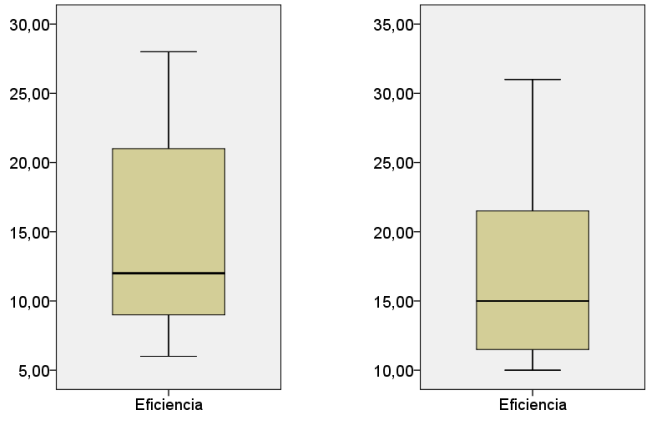
Variable	UC1: Estudiantes		UC2: Egresados y Profesionales	
	Media	Desv. Std	Media	Desv. Std
Efectividad	0.7974	0.10434	0.9436	0.07775
Eficiencia	14.3158	6.72518	17.4545	7.11848
Facilidad de uso percibida (PEOU)	4.1053	0.64418	4.1636	0.612
Utilidad percibida (PU)	4.2337	0.58483	4.17	0.36497
Intención de uso (ITU)	4.4042	0.56197	4.3336	0.53646

Tabla 6.20: Tabla de resumen de estadísticos descriptivos.

Los resultados globales nos han permitido concluir que MicroIoT ha mejorado el rendimiento de los participantes en prácticamente la totalidad de las estadísticas analizadas.

A. Eficiencia

La Figura 6.10 presenta los diagramas de caja para la variable Eficiencia para cada cuasi-experimento de lo cual se concluye que, bajo las condiciones de los experimentos, la aplicación de MicroIoT para las actividades “Análisis de requerimientos” y “Arquitectura de la solución” se requieren va desde los 6 hasta los 31 minutos aproximadamente.

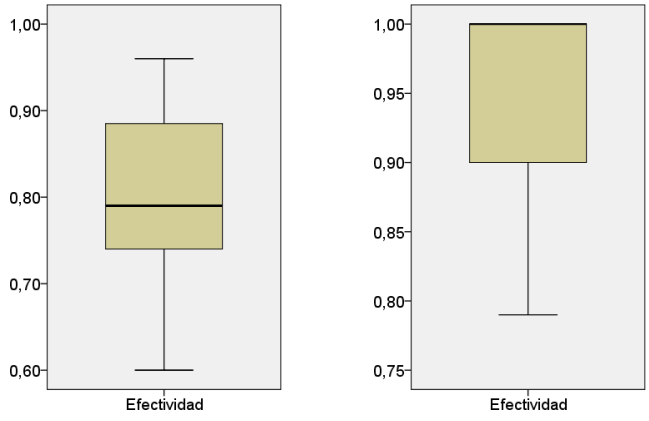


Eficiencia UC1 Eficiencia UC2

Figura 6.10: Diagrama de cajas de comparativas entre la eficiencia de los cuasi-experimentos efectuados.

B. Efectividad

La Figura 6.11 presenta los diagramas de caja para la variable Efectividad para cada cuasi-experimento de lo cual se concluye que, bajo las condiciones de los experimentos, los participantes alcanzaron entre un 60 % a 100 % de efectividad, con una media de 87 % de efectividad aproximadamente.



Efectividad UC1 Efectividad UC2

Figura 6.11: Diagrama de cajas de comparativas entre la efectividad de los cuasi-experimentos efectuados

C. Facilidad de uso percibida (PEOU)

La Tabla 6.21 presenta un resumen de las medias obtenidas para la facilidad de uso percibida en cada cuasi-experimento. La metodología propuesta, en general, fue percibida como fácil de usar en relación con el valor de prueba v=3.

Variable	**UC1: Estudiantes**	**UC2: Egresados y Profesionales**
Facilidad de uso Percibida (PEOU)	4.1053	4.1636

Tabla 6.21: Tabla resumen de medias de Facilidad de Uso Percibida (PEOU)

D. Utilidad percibida (PU)

La Tabla 6.22 presenta un resumen de las medias obtenidas para la utilidad percibida en cada cuasi-experimento. La metodología propuesta, en general, fue percibida como útil en relación con el valor de prueba v=3.

Variable	UC1: Estudiantes	UC2: Egresados y Profesionales
Utilidad percibida (PU)	4.2337	4.17

Tabla 6.22: Tabla resumen de medias de Utilidad Percibida (PU)

E. Intención de uso (ITU)

La Tabla 6.23 presenta un resumen de las medias obtenidas para la intención de uso percibida en cada cuasi-experimento. En general, los usuarios tienen la intención de utilizar la metodología en caso de requerir desarrollar una aplicación basada en microservicios para ambientes de IoT y AAL, esto se confirma ya que la intención reflejada es superior al valor de prueba v=3.

Variable	UC1: Estudiantes	UC2: Egresados y Profesionales
Intención de Uso (ITU)	4.4042	4.3336

Tabla 6.23: Tabla resumen de medias de Intención de Uso (ITU)

F. Resumen de las hipótesis

La Tabla 6.24 muestra un resumen de los resultados obtenidos en cada experimento individual.

Experimento	Tipo de participantes	Número de participantes	Hipótesis aceptadas	Hipótesis nulas rechazadas
UC1	Estudiantes de último año de la carrera de Ingeniería de Sistemas	19	$H1_1$, $H2_1$,$H3_1$, $H4_0$, $H5_0$, $H6_1$, $H7_0$, $H8_1$.	$H1_0$, $H2_0$,$H3_0$, $H6_0$, $H8_0$.
UC2	Egresados y profesionales de la carrera de Ingeniería de Sistemas	11	$H1_1$, $H2_1$,$H3_1$, $H4_0$, $H5_1$, $H6_0$, $H7_0$, $H8_1$.	$H1_0$, $H2_0$,$H3_0$, $H5_0$, $H8_0$.

Tabla 6.24: Tabla resumen de aceptación de hipótesis de los cuasi-experimentos.

Teniendo en cuenta los resultados obtenidos de los cuasi-experimentos, se concluye que para todos los participantes la metodología se percibe como útil, fácil de usar y se tiene la intención de uso en futuro en caso de requerir desarrollar aplicaciones basadas en microservicios en IoT y AAL.

Respecto a la efectividad, entre los dos cuasi-experimentos, se consiguió una media de alrededor del 87 %, lo cual significa que el proceso de diseño de dominio y adecuación de una arquitectura basada en microservicios se realiza de forma correcta. Por otra parte, la eficiencia de los participantes, en los dos cuasi-experimentos, es bastante aceptable (6 a 31 minutos). Esto podría deberse al desconocimiento de conceptos relacionados a microservicios e IoT por lo que los resultados podrían mejorar al profundizar en dichos aspectos.

G. Discusión

A continuación, se presentan las conclusiones globales que responden a las preguntas de investigación:

RQ1: ¿La metodología propuesta es percibida como fácil de usar y útil? De ser así, ¿las percepciones de los usuarios son el resultado de su rendimiento cuando utilizan el método para diseñar sistemas siguiendo una arquitectura basada en microservicios para Internet de las Cosas en Ambientes de Vida Asistidos?

Para responder esta pregunta es necesario evaluar las hipótesis H1, H2, H4 y H5 de los cuasi-experimentos. En los dos cuasi-experimentos UC1 y UC2 se rechazaron las hipótesis nulas H1 y H2 aceptando sus alternativas, esto implica que MicroIoT es percibida como una metodología útil y fácil de usar lo cual se corrobora con los resultados del cuasi-experimento ya que se obtuvo un 87 % de efectividad aproximadamente.

Respecto a si las percepciones del usuario son el resultado de su rendimiento al utilizar la metodología, se aceptó la hipótesis nula H4, lo que significa que la facilidad de uso percibida no puede verse determinada por la eficiencia de los participantes.

Respecto a la comprobación de si la utilidad percibida es resultado de la efectividad de los participantes, en el primer cuasi-experimento se aceptó la hipótesis nula H5, por lo que se consideró que la utilidad percibida no está determinada por la efectividad. Sin embargo, en el segundo cuasi-experimento se rechazó la hipótesis nula H5; esto significa que, la utilidad percibida no está determinada por la efectividad. La discrepancia entre los cuasi-experimentos puede deducirse que a medida que el cuasi-experimento se desarrolla en una mayor cantidad de tiempo se presencia una mayor dependencia de la utilidad percibida sobre la efectividad.

RQ2: ¿Existe una intención de uso de la metodología de software propuesta

en el futuro? De ser así, ¿tales intenciones de uso son el resultado de las percepciones de los participantes?

Respecto a la intención de uso futura de la metodología, se puede observar que, para los dos cuasi-experimentos, se rechazaron las hipótesis nulas H3 y se aceptaron sus alternativas, esto significa que los participantes sí tienen la intención de usar la metodología en el futuro cuando exista la necesidad de desarrollar aplicaciones basadas en microservicios para soluciones de IoT dentro de AAL.

Respecto a la incidencia de la facilidad de uso (PEOU) sobre la intención de uso (ITU), se observa que la hipótesis nula H7 fue rechazada en los dos cuasi-experimentos, esto significa que la intención de uso (ITU) sí está determinada por la facilidad de uso percibida (PEOU), es decir; los participantes consideran que el hecho de tener una arquitectura fácil de usar implica que esta pueda ser considerada para su uso en el futuro.

Finalmente, se puede observar que la hipótesis nula H8 fue rechazada y se aceptó su alternativa, es decir; los participantes consideran que la metodología es útil, éstos tendrían la intención de usarla en el futuro.

Los resultados globales del análisis de regresión son resumidos en las Figuras 6.12 y 6.13. Estos resultados constituyen una primera aproximación empírica de la evaluación de la arquitectura propuesta. Como trabajo futuro, se presenta la necesidad de investigar sobre la influencia de otras variables basadas en el rendimiento y la percepción para predecir la aceptación de la metodología propuesta en la práctica.

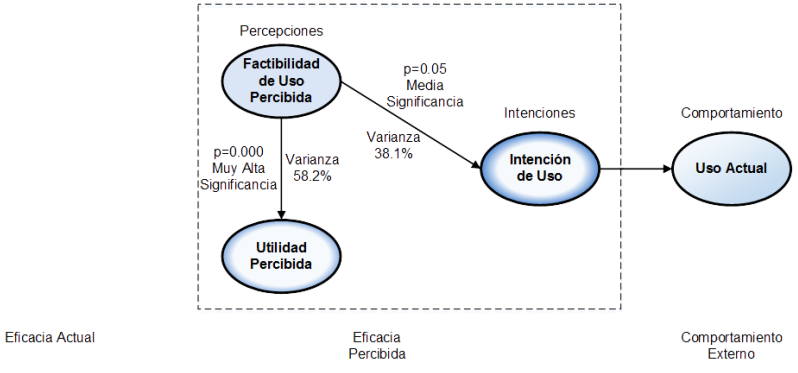


Figura 6.12: Conclusiones de la aplicación de MEM a la metodología propuesta - UC1.

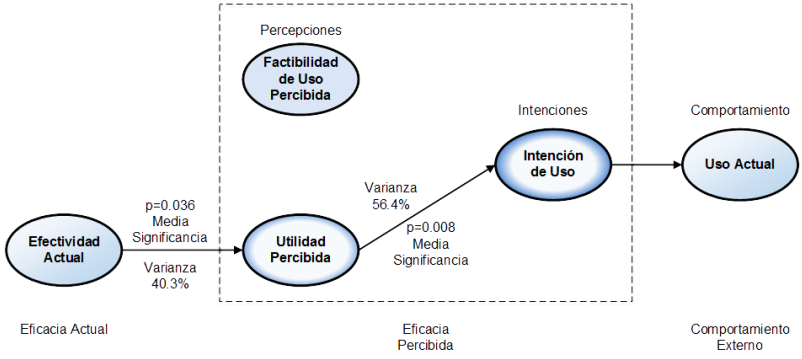


Figura 6.13: Conclusiones de la aplicación de MEM a la metodología propuesta - UC2.

6.6. Amenazas a la Validez

6.6.1. Validez interna

Las amenazas a la validez interna son relevantes en los estudios que intentan establecer relaciones causales. Las principales amenazas a la validez interna fueron: la experiencia de los participantes y los sesgos del autor.

Para reducir la amenaza relacionada con la experiencia de los participantes, previo al cuasi-experimento se dio una explicación sobre microservicios, y los conceptos aplicados en el cuasi-experimento (ver Anexo A.6). Además de ello, se preparó un ejemplo de entrenamiento, el cual muestra cada paso del proceso y provee a los usuarios un alto entendimiento tanto de la forma de desarrollar una arquitectura basada en microservicios aplicando nuestro método.

Tanto los sesgos del autor como las producidas por la entendibilidad del material fueron reducidos mediante la aplicación de un experimento piloto aplicado a dos sujetos de prueba para recibir una retroalimentación que ayude a mejorar la forma de desarrollar los cuasi-experimentos.

6.6.2. Validez externa

Esta se refiere a la habilidad para generalizar los resultados en diferentes contextos. Según (Cedillo, 2017), la principal amenaza a la validez externa es la representatividad de los resultados que puede verse afectada por el diseño de la evaluación, el contexto de participantes seleccionados y el tamaño y complejidad de las tareas experimentales.

El diseño de la evaluación puede tener un impacto en la generalización de los resultados debido a la complejidad de la metodología, la variedad de conceptos y elementos implicados. Se redujo este problema proponiendo un escenario bastante descriptivo que ayude a los sujetos a tener un panorama bastante claro de lo que se desea construir. Además, en la fase de entrenamiento se desarrolló un ejemplo en el que los evaluadores se centraron en el desarrollo de cada tarea a realizar, explicando cada fase conjuntamente con los conceptos requeridos, además de ello se respondió cualquier duda y cuestionamiento.

6.6.3. Validez del constructo

Las variables subjetivas están basadas en el *Method Evaluation Model* (MEM) (Moody, 2003), el cual, como se ha indicado, es un método muy bien conocido y validado empíricamente para la evaluación de tecnologías de la información. La principal amenaza es la confiabilidad del cuestionario (Cedillo, 2016). Por ello, se realizó una prueba de confiabilidad del alfa de Cronbach para cada conjunto de preguntas relacionadas a cada variable subjetiva. El umbral mínimo aceptado en el campo tecnológico es α= 0.70, en esta validación empírica, para la medición de la facilidad de uso percibida.

(PEOU) se obtuvo un α=0.822, para la medición de la utilidad percibida (PU) se obtuvo un α=0.747 y para la medición de la intención de uso (ITU) se obtuvo un α=0.742.

Como describe Cedillo (2017), las amenazas que afectan la validez de la conclusión se refieren a las conclusiones estadísticas, tales como la elección de los métodos estadísticos, y la elección del tamaño de la muestra.

Para controlar el riesgo de diferenciación de individuos, se seleccionó un grupo homogéneo de participantes. Respecto a la recolección de datos se aplicó el mismo procedimiento para cada experimento, además se garantizó que la misma función de medición se haya aplicado a cada variable dependiente.

6.6.4. Validez de la conclusión

Como describe (Cedillo, 2017), las amenazas que afectan la validez de la conclusión se refieren a las conclusiones estadísticas, tales como la elección de los métodos estadísticos, y la elección del tamaño de la muestra.

Para controlar el riesgo de diferenciación de individuos, se seleccionó un grupo homogéneo de participantes. Respecto a la recolección de datos se aplicó el mismo procedimiento para cada experimento, además se garantizó que la misma función de medición se haya aplicado a cada variable dependiente. A continuación, se explican los principales problemas que pueden poner en peligro la validez del cuasi-experimento.

6.7. Conclusiones

En este capítulo se presentó un cuasi-experimento y una réplica del mismo, con el objetivo de evaluar la eficacia percibida de un grupo de estudiantes, egresados y profesionales utilizando MicroIoT para desarrollar aplicaciones basadas en microservicios para soluciones IoT en AAL.

El método utilizado para validar la propuesta fue el *Method Evaluation Model* (MEM), el cual considera dos aspectos complementarios del éxito de un método: rendimiento actual y probabilidad de aceptación en la práctica. Para la aplicación de MEM se definió variables basadas en el rendimiento (eficiencia y efectividad) como factores de influencia para las variables basadas en la percepción (facilidad de uso percibida, utilidad percibida e intención de uso).

Los resultados obtenidos del análisis de los datos revelan que: (i) la mayoría de los participantes han encontrado que la metodología MicroIoT es fácil de usar y útil; (ii) la mayoría de los participantes indicaron consideran la utilización de MicroIoT en el futuro, en caso de ser necesario; (iii) el rendimiento de los participantes en los cuasi-experimentos determinó sus percepciones positivas; (iv) las percepciones determinan la intención de usar MicroIoT. En particular, los participantes han indicado en las preguntas abiertas del cuestionario, que les ha parecido útil el método ya que les ha guiado en la selección de métricas para medir cada cláusula del SLA.

Aunque los resultados son prometedores en cuanto a la utilidad del método para apoyar a los Arquitectos de Software durante las actividades: i) "Diseño de la entrega dirigida por el dominio" y ii) "Arquitectura de la solución", de la metodología MicroIoT, estos resultados son preliminares y deben ser considerados con cautela, ya que se basan en la evaluación de un conjunto reducido y resumido de las actividades planteadas en MicroIoT. Además, la metodología ha sido aplicada a usuarios sin experiencia previa en el desarrollo de aplicaciones de IoT para AAL basadas en microservicios. Por lo tanto, es necesario la realización de otros experimentos que consideren todas las actividades y tareas propuestas por MicroIoT aplicadas a sujetos experimentales con experiencia en Gestión de Proyectos y Gestores de Arquitecturas de Software, lo cual se vislumbra como trabajo futuro.

Edwin Cabrera Alvarado
Paola Cárdenas Cárdenas

Capítulo 7

Conclusiones y Trabajos Futuros

Este capítulo revisa los objetivos de la investigación, su nivel de cumplimiento y los principales hallazgos que han sido obtenidos de este trabajo. Finalmente, discute las contribuciones de esta investigación desde el punto de vista teórico, práctico y de las posibles líneas y oportunidades de investigación futura.

7.1. Conclusiones

A continuación se detalla la consecución de cada uno de los objetivos que han sido inicialmente planteados.

7.1.1. Objetivo general

El objetivo de esta tesis es *proponer una metodología para la creación de aplicaciones basadas en microservicios para soluciones de Internet de las Cosas (IoT) en Ambientes de Vida Asistidos (AAL - Ambient Assisted Living).*

A lo largo de este trabajo se ha planteado una metodología que abarca aspectos relacionados con la creación de aplicaciones de IoT para AAL basadas microservicios, considerando las actividades, desde la elicitación de requerimientos, hasta el despliegue de microservicios en un entorno de producción. Para ello, se han alineado las actividades de la metodología con el enfoque de desarrollo organizacional propuesto por DevOps.

Se ha definido un proceso de análisis de requerimientos basado en la priorización de los mismos. También, se ha planteado un proceso de identificación de microservicios a partir de un conjunto de requerimientos, Además, se ha establecido proceso de descomposición del personal permitiendo crear equipos multifuncionales cuya misión será encargarse de uno o varios microservicios en actividades de desarrollo, pruebas, integración y despliegue de los mismos. Uno de los aportes más relevantes es una arquitectura que constituye el nexo en el software (conformado por interfaces de usuario, y los microservicios) y el hardware (conformado por controladores, y dispositivos de hardware como sensores o actuadores) permitiendo al usuario conocer el esquema de la aplicación a desarrollar omitiendo aspectos técnicos.

Esta solución presenta los siguientes beneficios:

1. Tras una revisión sistemática, se determinó que no existen metodologías para el desarrollo de aplicaciones basadas en microservicios y orientadas a entornos de IoT y AAL, a pesar que la metodología se orienta a esas áreas, ésta puede escalar y orientarse a diferentes ámbitos de aplicación.

2. Ofrece una visión unificada de diferentes aspectos a aplicar al desarrollar microservicios tales como: Diseño Dirigido por Dominio o patrones de Gestión de Microservicios.

3. La metodología al estar basada en DevOps, la metodología ofrece un aseguramiento de la calidad al reducir la intervención humana durante la integración, despliegue y monitoreo de los microservicios. Esto se logra mediante la implementación de herramientas orientadas a cada fase de DevOps.

4. Dentro de la metodología se propone una arquitectura que integra los microservicios e IoT permitiendo a los usuarios escalar tener una visión de su aplicación previa a la codificación del mismo.

5. Existe una mayor interacción con el interesado ya que interviene diferentes actividades: i) Interviene en la elicitación de requerimientos para indicar los requerimientos del sistema, ii) Interviene para indicar los requerimientos de una entrega en particular, iv) Interviene para indicar que la entrega a desarrollar es la indicada, previo a su codificación y v) Luego del despliegue de la aplicación, el interesado ofrece una retroalimentación para el siguiente ciclo de la metodología.

6. La metodología, si bien considera aspectos característicos de AAL, puede ser extendida a diferentes áreas del conocimiento modificando los aspectos.

Para el desarrollo de aplicaciones basadas en microservicios, durante la fase de planificación de DevOps la metodología abarca diferentes técnicas de identificación, y gestión de microservicios y recursos humanos, es por ello que, el s centre su aporte durante la se centra en la fase de planificación de una entrega de software en el ciclo de DevOps, debido a que en esta fase inicia con la elicitación de requerimiento y comprende las diferentes actividades requeridas para obtener una arquitectura de un sistema basado en microservicios a partir de un problema determinado.

Finalmente, para lograr el objetivo general, se plantearon ciertos objetos específicos, que son analizados a continuación.

7.1.2. Objetivo específico 1

Llevar a cabo un estudio del arte acerca de las técnicas y estrategias actualmente utilizadas para el desarrollo e implementación de ambientes de vida asistidos.

Este objetivo ha sido cubierto totalmente, para ello, se desarrolló una revisión sistemática que permita dar una visión de que metodologías, soluciones y procesos relacionados a IoT y AAL y basados en servicios web. La revisión inició con un total de 1538 estudios analizados los cuales, a través de un proceso de selección aplicando diferentes criterios de extracción dieron como resultado 48 estudios aceptados.

Como resultado, revisión sistemática mostró que, a pesar de existir estudios que proponen aplicaciones y prototipos, estas no siguen una metodología debido a que no existe una metodología de desarrollo de aplicaciones para IoT y AAL. Por ello, la metodología propuesta puede ser considerada como una base para desarrollar aplicaciones escalables orientadas a IoT y AAL.

Otro resultado relevante de la metodología es que, a lo largo de los últimos 10 años el principal tipo de Arquitectura de Software fue la Arquitectura Orientada a Servicios (SOA), sin embargo; a partir de 2014 se evidencia una aumento de estudios relacionados a la Arquitectura Basada en Microservicios (MSA), por lo tanto; la metodología propuesta puede servir como base para aplicaciones investigaciones futuras en MSA debido a que adopta como núcleo a los microservicios y toda la teoría relacionada a la misma.

7.1.3. Objetivo específico 2

Identificar los requerimientos generales de aplicaciones basadas en Microservicios para soluciones de Internet de las Cosas en Ambientes de Vida Asistidos, establecer aquellos importantes e imprescindibles, incluyendo aspectos funcionales y no funcionales.

Este objetivo ha sido cubierto totalmente, ya que, previo a la definición de la metodología se realizó una revisión sistemática que permita identificar metodologías, soluciones, y procesos relacionados con IoT y AAL. Como resultado de este estudio empírico, se identificó: i) Los principales tipos de servicios orientados a la salud a la que puede ir dirigida una aplicación. ii) Principales entidades a las cuales puede ir dirigida la aplicación, tales como pacientes, cuidadores y profesionales de la salud. Estos criterios permiten identificar los requerimientos funcionales de AAL en una aplicación. Respecto a los requerimientos no funcionales, la revisión sistemática permitió identificar: i) Estándares de salud aplicados para el tratamiento de la información del paciente, ii) Los aspectos de calidad acordes a la ISO 25010 más utilizados. Estos criterios constituyen los requerimientos no funcionales a considerar al momento de desarrollar una aplicación utilizando la metodología propuesta.

7.1.4. Objetivo específico 3

Elaborar una metodología para la planificación de aplicaciones para ambientes asistidos dirigidos a personas pertenecientes a sectores vulnerables de la sociedad.

Este objetivo ha sido cubierto en su totalidad debido que la metodología, al tomar como base el enfoque organizacional DevOps también adopta como primera fase la planificación del sistema donde se abarca las siguientes actividades: i) Se realiza un análisis de requerimientos donde el interesado indica las necesidades y requerimientos que debe tener el sistema, ii) Posteriormente, se realiza un análisis de dominio donde se define las funcionalidades que tendrá la entrega y, a partir de ello se define los microservicios a desarrollar y se le asigna a un determinado equipo multifuncional, iii) Generar una arquitectura que presente los componentes que tienen la arquitectura y una arquitectura que facilite la gestión de microservicios, iv) Previo a la codificación, se realiza una última validación de la entrega.

7.1.5. Objetivo específico 4

Identificar tecnologías adecuadas para la construcción de la metodología, evaluarlas, compararlas y recomendarlas según los escenarios propuestos.

Este objetivo ha sido cubierto totalmente, a través del capítulo 2 que abarca el marco tecnológico, en el que se investigó acerca de las diferentes metodologías ágiles, considerando estudios comparativos y descriptivos y optando por el enfoque organizacional DevOps, además se consideró estudios que comparan

y analizan las herramientas existentes para la automatización de las fases de desarrollo, pruebas, despliegue y operaciones.

7.1.6. Objetivo específico 5

Evaluar la metodología desarrollada en un escenario puntual y real, con un estudio de caso o prueba de conceptos.

Este objetivo ha sido cubierto totalmente a través del Capítulo 6 donde se realizó 2 cuasi-experimentos a egresados y estudiantes de los últimos años de la carrera de Sistemas de la Universidad de Cuenca, y a Ingenieros en Sistemas que pertenecieron a esta institución. Las actividades abarcadas durante el cuasi-experimento son: "Diseño de entrega Dirigida por Dominio" y "Arquitectura del sistema o entrega". Esto se realizó A través de dichas actividades los sujetos experimentales alcanzaron a diseñar y establecer la arquitectura de la aplicación basada en microservicios a partir de un modelo de dominio planteado. Las actividades que involucran el desarrollo, pruebas, despliegue y operaciones no fueron consideradas durante dicho cuasi-experimento debido a que el proceso que abarcan dichas actividades involucran la codificación de la solución y la configuración de herramientas para la automatización de tareas, lo cual no forma parte de los objetivos de este proyecto de titulación.

7.2. Trabajos Futuros

Este trabajo de titulación no es el final de los esfuerzos de investigación en este área. Muchas actividades aún se pueden realizar para mejorar y ampliar la metodología aquí propuesta y el trabajo futuro irá en esa dirección. Los principales aspectos que se piensa abordar son analizados a continuación.

7.2.1. Con respecto a MicroIoT, la metodología propuesta.

Brindar pautas para el análisis de una mayor cantidad de requerimientos funcionales y no funcionales, para la actividad de "Diseño de la Entrega Dirigida por el Dominio", la cual tiene como principal resultado, microservicios definidos, en torno a un contexto y subdominio del problema analizado.

También se plantea como trabajo futuro considerar, además del diseño dirigido por dominio, otras técnicas para el diseño de microservicios, tales como el diseño dirigido por capacidades, entre otras.

Basar las soluciones de IoT para AAL, diseñadas con las Actividades y Tareas de MicroIoT, en una arquitectura más compleja que expanda el dominio del problema o los componentes de la arquitectura de software.

Con respecto a la arquitectura planteada, se considera añadir una capa de interacción de los elementos de hardware con los usuarios finales, considerando aspectos de interacción Hombre-Máquina para mejorar la experiencia de uso.

Debido a que MicroIoT orienta sus principales actividades a la planificación, como trabajo futuro, se pretende realizar un análisis con mayor detalle de las herramientas de automatización requeridas para las actividades de desarrollo y pruebas, despliegue, y operaciones.

7.2.2. Con respecto a la validación y mejora en las evaluaciones de la solución.

Se plantea evaluar todas las actividades que comprenden la metodología, dado que se han evaluado únicamente las actividades que no requieren de codificación, ni gestión y configuración de herramientas de automatización de actividades.

Réplicas de los estudios empíricos teniendo como sujetos a profesionales del área que hayan trabajado con arquitecturas de microservicios o involucrados en la industria y otros sectores que permitan validar la solución desde diferentes puntos de vista de usuarios.

Finalmente, se buscará evaluar la usabilidad/experiencia de usuario en cuanto a la aplicación de las actividades propuestas, a través de casos de estudio más complejos.

Edwin Cabrera Alvarado
Paola Cárdenas Cárdenas

Glosario

- **Agnóstico:** En este documento la palabra agnóstico se usa como una característica de los microservicios, debido a que no dependen de tecnologías a ni de un proceso de negocio concreto.
- **Agregado:** Un agregado es un artefacto de DDD, define un límite de coherencia alrededor de una o varias entidades, se compone de una sola entidad o un clúster de entidades y objetos de valor que deben permanecer transaccionalmente consistentes a lo largo de toda la vida del agregado. Una entidad exacta en un agregado es la raíz del agregado y por medio de ésta se lo puede identificar en el proceso de DDD.
- **Ambientes de Vida Asistidos (*Ambiente Assited Living - AAL*):** Comprende conceptos, productos y servicios interoperables que combinan las nuevas tecnologías de información y comunicación (TIC) y entornos sociales con el objetivo de mejorar y mejorar la calidad de vida de las personas en todas las etapas del ciclo de vida. AAL se preocupa principalmente por el individuo en su entorno inmediato al ofrecer interfaces fáciles de usar para todo tipo de equipos en el hogar y en el exterior, teniendo en cuenta que muchas personas mayores tienen deficiencias en la visión, la audición, la movilidad o destreza Pieper et al. (2011).
- **Autocontención:** El término autocontención se usa para describir que los servicios deben contener todo lo que necesitan para cumplir su tarea por sí mismos. Esto incluye no sólo su lógica comercial, sino también su front-end y back-end, así como también las bibliotecas requeridas (Butzin et al., 2016).
- **API:** Una interfaz de programación de aplicaciones (API) es un conjunto de funciones, procedimientos, métodos o clases que utilizan los programas informáticos para solicitar servicios del sistema operativo, bibliotecas de software u otro servicio, ya sea localmente o a través de una aplicación o servicio basado en la web (Bell et al., 2016).

- **Aplicación monolítica:** Una aplicación monolítica describe una aplicación de software de un solo nivel en la que la interfaz de usuario y el código de acceso a datos se combinan en un solo programa desde una sola plataforma. Una aplicación monolítica es autónoma e independiente de otras aplicaciones informáticas (Bell et al., 2016).

- **Arquitectura:** La arquitectura es la estructura de los componentes, sus interrelaciones y los principios y directrices que rigen su diseño y evolución en el tiempo (Bell et al., 2016).

- **Arquitectura de Microservicios (MSA):** La Arquitectura de Microservicios (MSA) es un estilo de arquitectura que define y crea sistemas mediante el uso de pequeños servicios independientes y autónomos que se alinean estrechamente con las actividades comerciales (Bell et al., 2016).

- **Arquitectura Orientada a Servicios (SOA):** SOA es un patrón arquitectónico en el diseño de software de computadora en el que los componentes de aplicaciones proporcionan servicios a otros compone (Bell et al., 2016).

- **Computación en la nube (*Cloud computing*)**: La computación en la nube es un modelo que permite acceso de red omnipresente, conveniente y bajo demanda a un grupo compartido de recursos informáticos configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que se pueden aprovisionar y liberar rápidamente con una administración mínima esfuerzo (Bell et al., 2016).

- **DevOps:** DevOps es un enfoque organizacional que abarca una cultura, movimiento o práctica que enfatiza la colaboración y comunicación de los desarrolladores de software y otros profesionales de la tecnología de la información (TI), mientras automatiza el proceso de entrega de software e infraestructura cambios.

- **Diseño Dirigido por Dominio (DDD):** El Diseño Dirigido por Dominio (DDD) proporciona un conjunto de principios y métodos para administrar la complejidad identificando dominios centrales y auxiliares y abordando Arquitectura, Desarrollo, Operaciones, Equipos, etc. dentro del Contexto Acotado de cada dominio (Bell et al., 2016).

- **Diseño orientado a objetos (OOD):** Object-Oriented Design (OOD) es la aplicación de una metodología orientada a objetos para el diseño de sistemas o aplicaciones informáticas (Bell et al., 2016).

- **Entidad:** Una entidad es un objeto con una identidad única que persiste en el tiempo, este identificador es único en el sistema y puede usarse para buscar la entidad o para recuperarla.
- **Implementación continua (CD):** La implementación continua (CD) puede considerarse una extensión de la integración continua, con el objetivo de minimizar el tiempo de entrega, el tiempo transcurrido entre el desarrollo de una nueva línea de código y el uso de este nuevo código por usuarios en vivo, en producción (Bell et al., 2016).
- **Integración continua (CI):** La integración continua (CI) es una práctica de desarrollo que requiere que los desarrolladores integren código en un repositorio compartido varias veces al día. Luego, cada registro se verifica mediante una compilación automática, lo que permite que los equipos detectan problemas con anticipación (Bell et al., 2016).
- **Internet de las cosas (IoT):** Es la red de objetos físicos (dispositivos, vehículos, edificios y otros elementos integrados con componentes electrónicos, software, sensores y conectividad de red) que permite que estos objetos recopilen e intercambien datos (Bell et al., 2016).
- **Microservicio:** Un microservicio individual es un servicio que se implementa con un único propósito, que es autónomo e independiente de otras instancias y servicios. Un microservicio es un elemento arquitectónico primario de una arquitectura de microservicios (Bell et al., 2016).
- **Objeto de Valor:** Un objeto de valor no tiene identidad. Se define únicamente mediante los valores de sus atributos. Los objetos de valor también son inmutables.
- **Resiliencia** La cual consiste en la capacidad de resistencia ante los riesgos, regresando a un estado original, siempre y cuando el impacto no exceda el límite de tolerancia, y sin dejar de cumplir con sus objetivos misionales (Jaramillo and Palacios).
- **Resistencia:** La capacidad de recuperación de una aplicación es la capacidad de una aplicación para reaccionar a los problemas en uno de sus componentes y aún así proporcionar el mejor servicio posible (Bell et al., 2016)

Apéndice A

Anexos

Edwin Cabrera Alvarado
Paola Cárdenas Cárdenas

A.1. Lista de estudios seleccionados

Num	Fuente	Título	Año
1	ScienceDirect	Towards fog-driven IoT eHealth: Promises and challenges of IoT in medicine and healthcare	2018
2	ScienceDirect	A lightweight framework for transparent cross platform communication of controller data in ambient assisted living environments	2015
3	ScienceDirect	A Web Platform for Interconnecting Body Sensors and Improving Health Care	2014
4	ScienceDirect	Risk driven Smart Home resource management using cloud services	2014
5	ScienceDirect	Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: A fog computing approach	2018
6	ScienceDirect	Consume: A privacy-preserving authorisation and authentication service for connecting with health and wellbeing APIs	2018
7	ScienceDirect	Smart Home Assistant for Ambient Assisted Living of Elderly People with Dementia	2017
8	IEEEXplore	Anticipating Health Hazards through an Ontology-Based, IoT Domotic Environment	2012
9	IEEEXplore	Integrating IoT and IoS with a Component-Based approach*	2010
10	IEEEXplore	An IoT-inspired Cloud-based Web Service Architecture for e-Health Applications	2016
11	IEEEXplore	Improving life quality for the elderly through the Social Internet of Things (SIoT)	2017
12	IEEEXplore	Microservices Model in WoO based IoT Platform for Depressive Disorder Assistance	2017
13	SpringerLink	Using SOA for a Combined Telecare and Telehealth Platform for Monitoring of Elderly People	2012
14	SpringerLink	IoT-Based Health Monitoring System for Active and Assisted Living	2017
15	SpringerLink	W2T Framework Based U-Pillbox System Towards U-Healthcare for the Elderly	2016
16	SpringerLink	Health Monitor: An Intelligent Platform for the Monitorization of Patients of Chronic Diseases	2018

17	SpringerLink	eWALL: An Open-Source Cloud-Based eHealth Platform for Creating Home Caring Environments for Older Adults Living with Chronic Diseases or Frailty	2017
18	SpringerLink	From Patient Information Services to Patient Guidance Services-The iCare Approach	2013
19	SpringerLink	Situated Agents and Humans in Social Interaction for Elderly Healthcare: From Coaalas to AVICENA	2015
20	SpringerLink	Data-Driven Smart Home System for Elderly People Based on Web Technologies	2016
21	ACM	A novel and secure IoT based cloud centric architecture to perform predictive analysis of users activities in sustainable health centres	2016
22	ACM	A Service-Oriented Mobile Cloud Middleware Framework for Provisioning Mobile Sensing as a Service	2015
23	ACM	Smart Spaces and Smart Objects interoperability Architecture (S3OiA)	2012
24	ACM	COLLECT: COLLaborativE ConText-aware service oriented architecture for intelligent decision-making in the Internet of Things	2017
25	ACM	Enabling the IoT paradigm in e-health solutions through the VIRTUS middleware	2012
26	ACM	Integrating IoT and IoS with a Component-Based approach*	2010
27	ACM	Mobile Healthcare Infrastructure for Home and Small Clinic	2012
28	Healthcom	Platform for building eHealth streaming services	2013
29	Healthcom	Self-adaptive Middleware for ubiquitous Medical Device Integration	2014
30	Healthcom	A Dynamic Cloud Computing Platform for eHealth Systems	2015
31	Healthcom	A conceptual model for integrating social and health care services at home: the H@H project	2016
32	Healthcom	Distributed Performance Management of Internet of Things as a Service for Caregivers	2017

33	Healthcom	SimpleHealth – a Mobile Cloud Platform to Support Lightweight Mobile Health Applications for Low-end Cellphones	2017
34	Healthcom	INTERACCT: Remote Data Entry System with Game-Elements for young Leukaemia Patients	2015
35	Healthcom	A Framework for Customizing the Mobile and Remote Monitoring of Patients with Chronic Diseases	2014
36	Healthcom	RESTful Services for an Innovative E-Health Infrastructure: A Real Case Study	2014
37	Healthcom	A Mobile Volunteered Geographic Information Management Platform for Rural Health Informatics	2015
38	Healthcom	Enhancing eHealth Smart Applications: A FogEnabled Approach	2015
39	Healthcom	IReHMo: An Efficient IoT-Based Remote Health Monitoring System for Smart Regions	2015
40	Healthcom	Processes' Optimization Tools for Web-service Oriented Organizations' Resources – An eHealth Application	2015
41	Healthcom	Rapid Delivery e-Health Service (RDeHS) Platform	2016
42	Healthcom	Internet of the Body and Cognitive Companion	2017
43	Healthcom	Remote Patient Health Monitoring Cloud Brokering Services	2017
44	ICSOC	Context as a Service: Realizing Internet of Things-Aware Processes for the Independent Living of the Elderly	2016
45	Qshine	Detailed Dominant Approach Cloud Computing	2013
46	SCC	A Personal Healthcare System with Inference-as-a-Service	2015
47	SOCA	Dynamic Homecare Service Provisioning Architecture	2011
48	SOCA	A Service-Oriented Architecture for Web Applications in e-mental health two case studies	2015

Tabla A.1: Lista de estudios seleccionados para la revisión sistemática

A.2. Resultados de Revisión Sistemática

RQ1: ¿Cuáles son las soluciones propuestas en las aplicaciones existentes en las aplicaciones existentes?			
Criterio	**Solución planteada**	**Cuenta**	**%**
Solución planteada	Metodología	1	2 %
	Arquitectura	19	40 %
	Framework	14	29 %
	Prototipo	16	33 %
	Aplicación de software	28	58 %
	Dispositivo de hardware	14	29 %
	Otros	4	8 %
Ambiente de despliegue	Web	23	48 %
	Aplicación móvil	22	46 %
	Escritorio	6	13 %
	Aplicación embebida	9	19 %
	No especifica	6	13 %
Modelo de Servicio	Infraestructura como servicio (IaaS)	7	15 %
	Plataforma como servicio (PaaS)	9	19 %
	Software como servicio (SaaS)	29	60 %
	IoT como servicio (IoTaas)	8	17 %
	Context as a Service (CaaS)	3	6 %
	MWaaS (Involucra SaaS y PaaS) Middleware	5	10 %
	Otros	3	6 %
RQ2: ¿Qué aspectos de Ingeniería de Software se consideran al momento de desarrollar aplicaciones empleando microservicios en IoT y AAL?			
Criterio	**Solución planteada**	**Cuenta**	**%**
Tipo de Arquitectura de software	Modelo Vista Controlador. (MVC)	4	8 %
	Cliente-servidor	11	23 %
	Arquitectura de Microservicios.(MSA)	7	15 %
	Arquitectura publish/subscribe OSGi	7	15 %
	Arquitectura orientada a servicios (SOA)	26	54 %

Tipo de Arquitectura de software	Pipeline	1	2 %
	Arquitectura dirigida por modelos (MDA)	4	8 %
	La arquitectura basada en eventos (EDA)	1	2 %
	Arquitectura Propia	2	4 %
	Otras	4	8 %
Plataforma de despliegue	Hosting	8	17 %
	Cloud computing	23	48 %
	Fog computing	4	8 %
	Edge computing	5	10 %
	Localmente	5	10 %
	No especifica	15	31 %
Modelo de Despliegue Cloud	Pública	5	10 %
	Privada	4	8 %
	Hibrida	4	8 %
	No especifica	38	79 %
Fases de Entrega Continua de software que se consideran	Integración Continua	17	35 %
	Pruebas	23	48 %
	Producción	22	46 %
	No Aplica	21	44 %
Metodología de desarrollo de software Aplicada	Ágiles	5	10 %
	Cascada	6	13 %
	Evolutivas	6	13 %
	DevOps	8	17 %
	No Especifica	29	60 %
Aspectos de calidad según ISO 25010	Eficiencia	13	27 %
	Usabilidad	20	42 %
	Fiabilidad	8	17 %
	Disponibilidad	14	29 %
	Seguridad	25	52 %
	Mantenibilidad	10	21 %
	Interoperabilidad	17	35 %
	Otros	3	6 %
RQ3: ¿Cuáles son las tecnologías utilizadas en la creación de aplicaciones para soluciones de IoT en AAL?			
Criterio	**Solución planteada**	**Cuenta**	**%**
Bases de datos	Relacional	14	29 %
	No relacional	11	23 %

Bases de datos	No especifica	24	50 %
Formato de salida de datos	XML	17	35 %
	JSON	10	21 %
	HTML	3	6 %
	No especifica	21	44 %
Estandares de comunicacion	Z-wave	1	2 %
	KNX	1	2 %
	UPnP	1	2 %
	Java Message Service JMS	1	2 %
	No especifica	45	94 %
Conjuntos de estándares y sistemas de conceptos orientados a la salud	HL7 Reference Information Model	2	4 %
	HL7 Electronic Health Record	4	8 %
	X73	1	2 %
	ISO 13940 (ContSys)	1	2 %
	PQRST	1	2 %
	No especifica	41	85 %
Tipo de servicio web proporcionado	REST	20	42 %
	SOAP-WS	4	8 %
	No especifica	25	52 %
Protocolos de comunicación (TCP-IP)	XMPP	3	6 %
	MQTT	2	4 %
	CoAP	2	4 %
	AMQP	1	2 %
	HTTPS	14	29 %
	No especifica	31	65 %
Especifica herramientas	Si	29	60 %
	No	19	40 %
Herramientas de Gestión de Contenedores	Docker	2	4 %
	Knopflerfish (Manager Contenedores)	1	2 %
Middlewares	RabbitMQ	1	2 %
	VIRTUS	1	2 %
	OpenHAB	1	2 %
	Hydra Middleware	1	2 %
Herramientas de Desarrollo de SW	Java EE	3	6 %
	Apache Struts	1	2 %
	Node.js	4	8 %

Servidores de Aplicaciones	Glassfish server	1	2 %
	Apache Tomcat	5	10 %
Frameworks	Spring	2	4 %
	Ruby on Rails	1	2 %
	Django	1	2 %
	ASP.NET	1	2 %
	Apache Felix	1	2 %
Herramientas Cloud Management	OpenStack	2	4 %
	Microsoft Azure IoT Hub	4	8 %
	IBM Bluemix	1	2 %
	Google App Engine	1	2 %
	Amazon Web Services Platform	4	8 %
Herramientas de Gestión de datos	RoQua	1	2 %
	IBM Watson	1	2 %
	Websphere Lombardi Edition	1	2 %
	Websphere ILOG rules	1	2 %
	Apache Kafka	1	2 %
	Hibernate	1	2 %
Herramientas estándar	Web Processing Service(WPS)	4	8 %
	Web Feature Service o WFS	1	2 %
	Web Socket	2	4 %
	DPWS	1	2 %
Lenguaje de Programación (Implementación de Microservicios)	C++	2	4 %
	Javascript	7	15 %
	PHP	1	2 %
	Java	17	35 %
	R	1	2 %
	Python Web	1	2 %
	No especifica	24	50 %
RQ4: ¿Qué necesidades o problemas son abordados a través de las soluciones desarrolladas?			
Criterio	**Solución planteada**	**Cuenta**	**%**
Orientación	Pacientes	42	88 %
	Cuidadores	19	40 %
	Familiares	12	25 %
	Profesionales de la Salud	20	42 %
	Administradores de eHealth system	6	13 %

Orientación	Sistemas con Inteligencia Artificial	9	19 %
	Otros	4	8 %
Tipo de servicio de salud	Asistencia para medicación	10	21 %
	Asistencia para personas mayores	14	29 %
	Asistencia de estilo de vida	19	40 %
	Manejo de enfermedades crónicas	13	27 %
	Cuidados paliativos	10	21 %
	Gestión de datos de salud	6	13 %
	Industria	18	38 %
	Servicios de rehabilitación	5	10 %
	Monitoreo de la salud	31	65 %
Dominio de cuidado de la salud identificado	Home Health	27	56 %
	e-Health	16	33 %
	M-Health(Health mobile)	19	40 %
	Ubiquitous Health	12	25 %
	Hospital Management.	10	21 %
	WSN (Wireless sensor networks)	21	44 %
RQ5: ¿Cómo se está desarrollando actualmente la investigación de esta área?			
Criterio	**Solución planteada**	**Cuenta**	**%**
Campos de aplicación de los estudios	Industria	16	33 %
	Académico	42	88 %
Métodos de validación	Experimentos controlados	5	10 %
	Cuasi experimentos	5	10 %
	Pruebas de conceptos	16	33 %
	Casos de estudio	28	58 %
Tipo de estudio	Nuevo	32	67 %
	Extensión	16	33 %

Tabla A.2: Resultados revisión sistemática

A.3. Símbolo, Términos y Descripción de SPEM 2.0

En este anexo se describe cómo se utiliza la notación SPEM 2 (*Software & System Process Engineering Meta-model Specification V2.0*), propuesta por el grupo *Object Management Group*, que provee un marco formal para la definición de procesos de desarrollo de sistemas y de software, así como también para definir y describir sus elementos. El objetivo de su uso en este trabajo de titulación, es el proveer una definición detallada del proceso que sigue Micro-IoT, la metodología propuesta, proveyendo así, una manera simple y fácil de entender.

De acuerdo con Cedillo (2017), SPEM es parte de la Ingeniería de Procesos de Software (SEP), dedicándose a la definición, implementación, medición y mejora de los procesos de Ingeniería del Software. Está basada en MOF (Meta Object Facility), que es un estándar de la OMG, siendo SPEM a los procesos software lo mismo que UML a los sistemas software.

SPEM constituye un meta-modelo que permite definir modelos de procesos de Ingeniería del Software y de Ingeniería de sistemas. Este detalla los elementos mínimos necesarios que permitan definir dichos procesos sin añadir aspectos del dominio particular. La idea central de SPEM 2 está basada en tres elementos básicos: rol, producto de trabajo y tarea, éstos elementos se describen en la Tabla A.3.

Actividad o Tarea	Esfuerzo a realizar.
Rol	Quién realiza el esfuerzo.
Productos de trabajo	Entradas que se utilizan en la tarea y salidas que éstos producen.

Tabla A.3: Elementos básicos de los diagramas SPEM.

Por medio de estos elementos se puede especificar "Quien (rol) realiza qué (actividad) o qué (tarea) para desde unas entradas conseguir unas salidas (productos de trabajo)".
En este documento se consideran siete actividades principales y cada una de éstas abarca una o várias tareas, es por ello que se presenta un diagrama SPEM a nivel de actividades que se representan con el elemento "Tarea"de SPEM 2.0 (ver Anexo A.4) y a nivel de tareas a lo largo de este trabajo de titulación.

De acuerdo con Cedillo (2017) SPEM 2 distingue dos etapas cuando se define un proceso:

1. Definición de los elementos de contenido que son los elementos primarios

o constructores básicos, sus relaciones se ilustran en la Figura A.1.

- Roles
- Tareas
- Productos de Trabajo
- Categorías

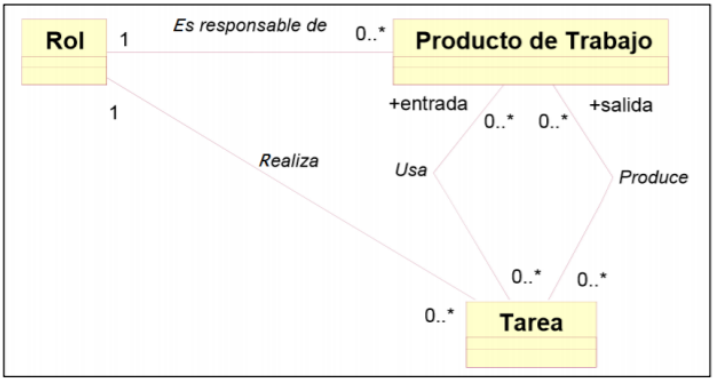


Figura A.1: Metamodelo SPEM (Fuente: (Cedillo, 2014)).

Ventajas de Utilizar SPEM

- Se puede disponer de modelos de Procesos de software en formato procesable por computador.
- Facilita la comprensión y comunicación entre las personas puesto que propicia un conocimiento homogéneo.
- Da soporte a la mejora de procesos.
- Da soporte a la gestión de procesos.
- Guía la automatización de procesos y da soporte para la ejecución automática.

A continuación, en la Figura A.2, se describe el subconjunto de primitivas de modelado que son las más comúnmente utilizadas a la hora de definir un proceso:

Icono	Nombre	Descripción
	Definición de Rol	Conjunto de habilidades, competencias y responsabilidades relacionadas, de un individuo o de un grupo.
	Definición de Tarea	Unidad de trabajo asignable y gestionable, identificando el trabajo que se ejecuta por los roles. Puede dividirse en varios pasos.
	Definición de Producto de trabajo	Producto usado o producido por las *Tareas*. Existen dos tipos de productos: *Artefacto* de naturaleza tangible (modelo, documento, código, archivos, etc.) y *Entregable* para empaquetar productos con fines de entrega a un cliente interno o externo. Se pueden asociar entre ellos mediante relaciones de agregación, composición e impacto.
	Categoría	Clasificación de elementos como *Tareas, Roles* y *Productos* en base a los criterios que desee el ingeniero de procesos. Existen diversos tipos de categorías: *Conjunto de Roles* (para *Roles*), *Disciplina* (para *Tareas*), *Dominio* (para *Productos*),
	Guías	Información adicional relacionada con otros elementos. Los sub-tipos de guías pueden ser (entre otros): *Activo Reutilizable, Directriz, Documentación, plantillas*.
	Uso de Rol	Representación del r*ol* que lleva a cabo una *Tarea* o *Actividad* dentro de un proceso determinado. Hace referencia a una Definición de Rol (elemento de Contenido).
	Uso de Tarea	Representación de una *tarea* atómica dentro de un proceso determinado. Hace referencia a una Definición de Tarea (elemento de Contenido).
	Uso de Producto de Trabajo	Representación de un *Producto de Trabajo* de entrada o salida, relacionado con una *Actividad* o *Tarea*. Hace referencia a una Definición de un Producto de Trabajo (elemento de Contenido)
	Actividad	Representación de un conjunto de *Tareas* que se ejecutan dentro del proceso, junto con sus *Roles* y *Productos* asociados. Si únicamente se quiere representar una agrupación de tareas, se puede usar los elemento Actividad o Fase (incluido por retro-compatibilidad y más empleado en tareas de desarrollo), o bien si es un conjunto de tareas que se repite un determinado número de veces, se puede usar el elemento Iteración.
	Fase	
	Iteración	
	Paquete de Proceso	Representación de un paquete agrupando todos los elementos del proceso

Figura A.2: Simbología SPEM (Fuente: (Cedillo, 2014)).

A.4. Diagrama de procesos de metodología MicroIoT

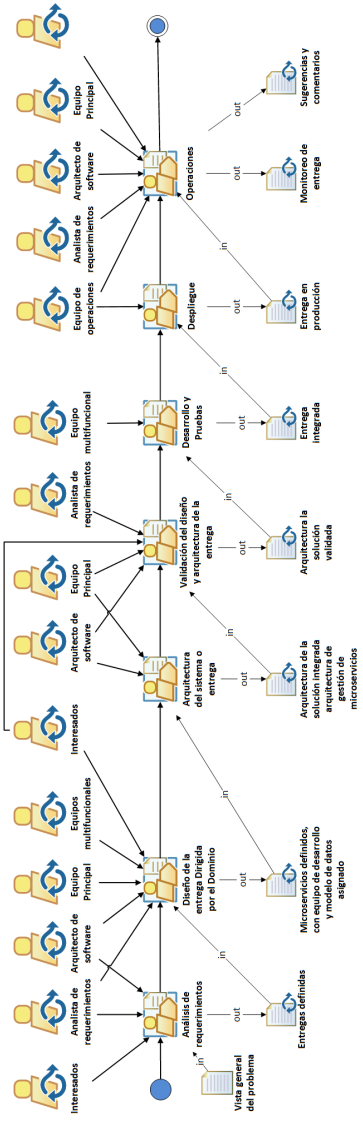


Figura A.3: Estructura del trabajo de titulación (Fuente: Elaboración propia).

A.5. Plantillas utilizadas durante la metodología

<table>
<tr><th colspan="2">Características Generales de AAL</th></tr>
<tr><td>Tipo o tipos de servicio de salud que abarca el sistema.</td><td>Orientación del sistema (usuarios finales)</td></tr>
<tr><td>Se sugiere las siguientes opciones:
- Medicación.
- Asistencia para adultos mayores.
- Asistencia de estilo de vida.
- Manejo de enfermedades crónicas.
- Vigilancia
- Otros</td><td>Se sugiere las siguientes opciones:
- Paciente
- Cuidador o Familiar
- Profesionales de la salud
- Otros</td></tr>
<tr><td>Requerimientos de Hardware (Dispositivos de Internet de las Cosas disponibles o necesarios)</td><td>Ambiente en el que opera el sistema</td></tr>
<tr><td>Se sugiere las siguientes opciones:
- Dispositivos controladores
- Dispositivos sensores
- Dispositivos actuadores
- Dispositivos inteligentes con sensores, actuadores y controladores integrados
- Otros</td><td>Se sugiere las siguientes opciones:
- Hogar
- Hospital
- Ubiquitous Health
- Otro</td></tr>
<tr><td colspan="2">Plataforma de despliegue de la o las aplicaciones del sistema
(para cada plataforma de despliegue de la o las aplicaciones es importante considerar a quién está orientada o quién será el usuario final de esa aplicación).
Se sugiere las siguientes opciones
- Aplicación Web
- Aplicación Móvil
- Aplicación de Escritorio
- Otro</td></tr>
</table>

Tabla A.4: Características Generales de AAL

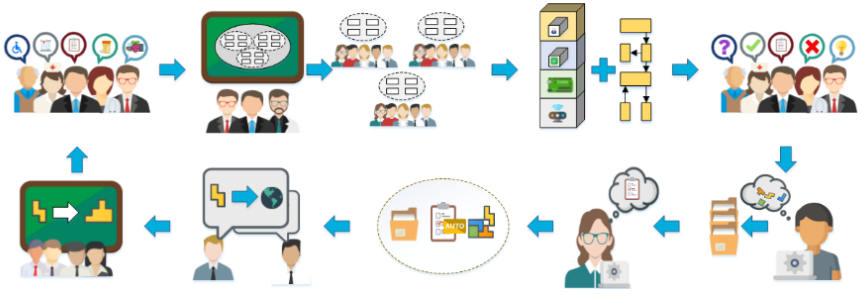
Aspectos de Calidad de Software Identificados	
Aspectos de calidad según ISO 25010	**Estándares de Salud**
Seleccionar los aspectos de calidad requeridos - Seguridad. - Usabilidad. - Interoperabilidad. - Eficiencia. - Disponibilidad. - Mantenibilidad. - Autenticidad. - Otro.	Seleccionar el estándar de salud a aplicar - HL7 *Reference Information Model* - HL7 *Electronic Health Record* - ISO 13940 (*ContSys*) - Otro

Tabla A.5: Requerimientos de calidad generales en AAL.

A.6. Guía de Cuasi-experimento

Micro IoT

Metodología para la creación de aplicaciones basadas en Microservicios para soluciones de Internet de las Cosas en Ambientes de Vida Asistida.



A.6.1. Presentación de la Metodología

En la documentación adjunta, en el Anexo 1, se encuentra la descripción de la metodología MicroIoT en la que se pueden consultar los conceptos necesarios para el entendimiento del vocablo y detalles del procedimiento durante la realización del presente ejercicio.

A.6.2. Objetivos de la evaluación

La presente tiene como objetivo medir la efectividad y la eficiencia con la que un Arquitecto/a de software, Ingeniero/a de Sistemas, o Desarrolladores de Software pueden entender la metodología propuesta y realizar las actividades: 2) Diseño de la entrega Dirigida por el Dominio y 3) Arquitectura de la solución, planteados en la primera etapa de la metodología, los cuales permiten obtener una aplicación o solución de Internet de las Cosas en Ambientes de Vida Asistida. Esta evaluación también pretende determinar la facilidad y la intención de uso futura, en base a las respuestas dadas a un cuestionario presentado luego de la aplicación de los pasos de la metodología.

Debido a que la actividad 1) Análisis de requerimientos se entrega resuelta al sujeto experimental debido a que involucra un proceso de elicitación que podría variar el resultado de las otras actividades evaluadas ya que podrían darse diferentes interpretaciones del problema. Por otra parte, las actividades 4) Validación del diseño y arquitectura de la entrega, 5) Desarrollo y Pruebas y 6) Despliegue, y 7) Operaciones, de la metodología no se consideran en la evaluación debido a que para evaluar dichos pasos, el sujeto experimental requiere de conocimientos profundos de la aplicación de patrones involucrados en la Arquitectura de Microservicios y su codificación, así como de las herramientas tecnológicas disponibles.

A.6.3. Procedimiento de la evaluación

Se plantea un escenario con una problemática de la vida real, se necesita encontrar una solución de Internet de las Cosas en Ambientes de Vida Asistida, realizando las actividades 1 y 2 que permiten diseñar la aplicación y establecer la arquitectura de la aplicación. El proceso se explica a continuación y está alineado a lo propuesto en la metodología como se puede confirmar en el Anexo 1. En la figura 1 se muestra la estructura del documento para la resolución del ejercicio planteado.

Una vez registrados sus datos personales y la hora (inicio de creación de la solución), en la sección (a) del documento de evaluación. se procede con la aplicación de las primeras actividades de la metodología. ***No olvide colocar***

la hora de inicio y la hora de fin en cada paso de la actividad 1 y en la actividad 2.

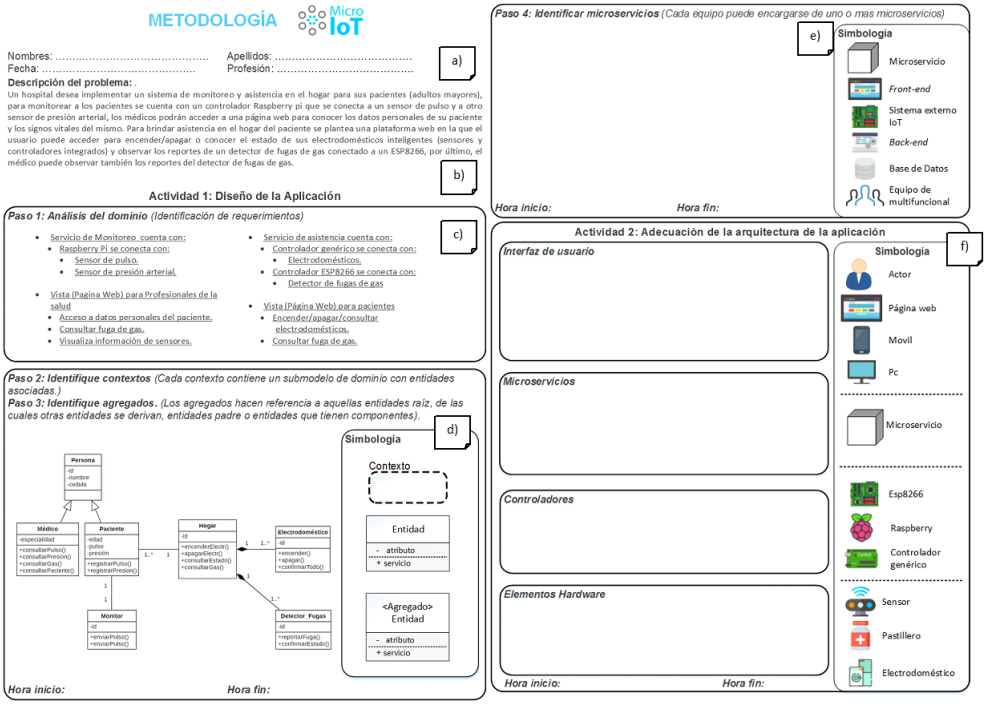


Figura 1: Documento de resolución del ejercicio propuesto.(a) Datos personales y generales, (b) Enunciado del ejercicio, (c) Análisis de dominio, (d) Definir contextos y agregados, (e) Identificación de requerimientos y sus componentes, y (f) adecuación de la arquitectura de la aplicación. (Fuente: Elaboración propia).

El diseño de la aplicación consta de 4 pasos,

1. **Actividad 1: Diseño de la aplicación.**

   a) ***Paso 1 (Análisis de Dominio):*** consiste en la elicitación de requerimientos en alto nivel, la cual permite identificar tanto los requerimientos como los objetivos del proyecto. Estos estarán ubicados en la sección (c) "Lista de Requerimientos" . Para facilitar la actividad los requerimientos ya están previamente identificados

en base al enunciado de ejemplo que se presenta en la sección (b) del documento de evaluación. (Figura 1). En la figura 2 se muestra el resultado del análisis del dominio del enunciado siguiente, mismo que se usará a lo largo de la explicación del procedimiento de evaluación:

Descripción del problema: .
Un hospital desea automatizar muchas de sus tareas para proporcionar un servicio de vigilancia y medicación controlada a sus pacientes, para la vigilancia cada paciente contará con un controlador genérico que controla un sensor de temperatura corporal y un detector de caídas, las enfermeras del hospital deben tener acceso a una página web que les permita ver la información del paciente y la información de los 2 sensores mencionados anteriormente. Para la medicación controlada se cuenta con un raspberry que envía la información recolectada por el pastillero de cada paciente (dosis, horario) a la nube. Solamente los médicos podrán tener acceso a la información del pastillero, es decir; si se ha cumplido con el horario de medicación y cuál fue la dosis administrada. Además el médico también puede consultar la temperatura del paciente y el registro del detector de caídas a través de una aplicación de escritorio.

Actividad 1: Diseño de la Aplicación

Paso 1: Análisis del dominio *(Identificación de requerimientos)*

- Servicio de vigilancia cuenta con:
  - Controlador genérico se conecta con:
    - Sensor de temperatura corporal.
    - Sensor detector de caídas.
- Vista (Pagina Web) para enfermeras
  - Visualiza información del paciente.
  - Visualiza información de sensores.
- Servicio de monitoreo cuenta con:
  - Controlador Raspberry pi se conecta con:
    - Un pastillero que registra la dosis y el horario del medicamento.
- Vista (Aplicación de escritorio) para los médicos
  - Visualiza información del pastillero.
  - Visualiza información del paciente.
  - Visualiza información de los sensores.

Figura 2: Lista de requerimientos del enunciado de ejemplo. (Fuente: Elaboración propia).

b) ***Paso 2 (Delimitación de contextos):*** Un contexto delimitado es simplemente el límite dentro de un dominio donde se aplica un modelo de dominio en particular. En el presente proceso de evaluación se pretende que los sujetos experimentales definan los contextos y escriban el nombre del contexto en el exterior de la línea punteada. En la Figura 3 se muestra la aplicación del paso 2 para el enunciado de ejemplo.

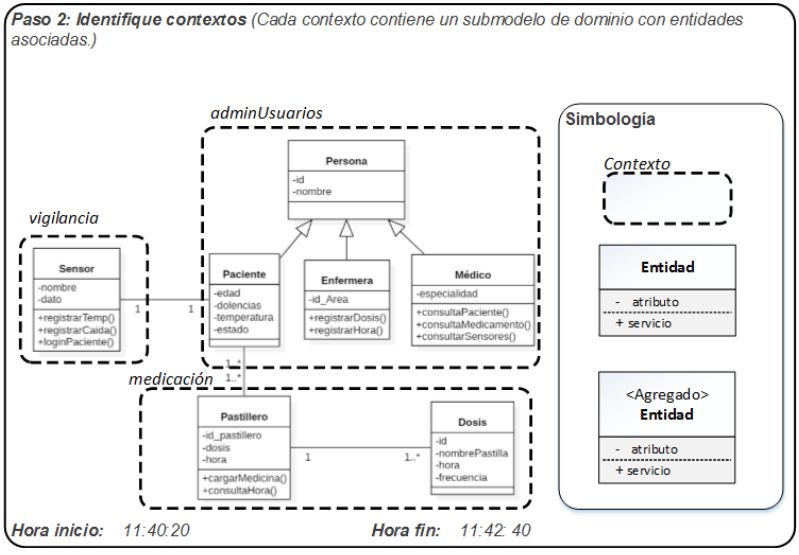


Figura 3: Delimitación de contextos del enunciado de ejemplo. (Fuente: Elaboración propia).

c) ***Paso 3 (Identificar agregados):*** Los agregados hacen referencia a aquellas entidades raíz, de las cuales otras entidades se derivan (entidades padre o entidades que tienen componentes), un agregado también se identifica por el tiempo de vida de la entidad (si una entidad raíz desaparece sus componentes también), debe colocar la palabra <**Agregado**>, encima del nombre de la entidad que considere es un agregado. En la figura 4 se muestra el resultado del paso 3, sugerido para el enunciado de ejemplo.

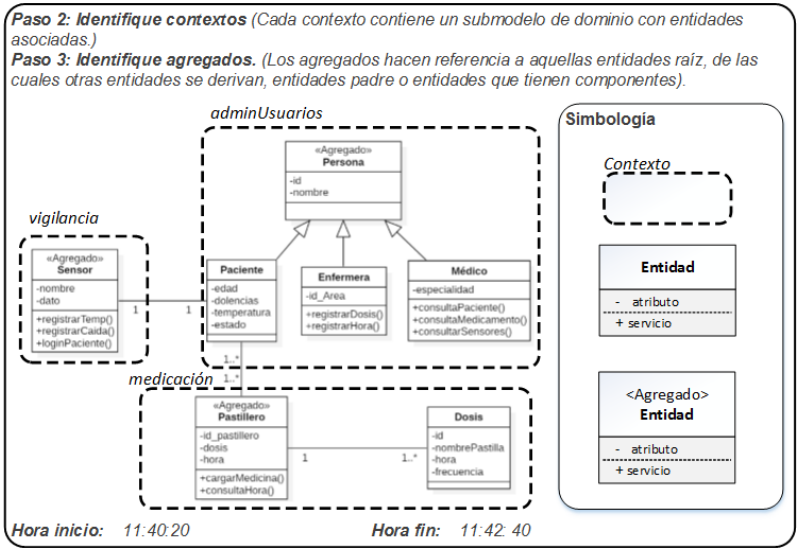


Figura 4: Definición de entidades agregados y servicios para el enunciado de ejemplo. (Fuente: Elaboración propia).

d) ***Paso 4 (Identificación y Validación de microservicios):*** En este paso se puede usar el principio fundamental propuesto: *"Un microservicio no debe ser menor que un agregado ni mayor que un contexto delimitado. Los agregados suelen ser buenos candidatos para microservicios"*. En la figura 5 se observa la aplicación del paso 5 para el enunciado de ejemplo.

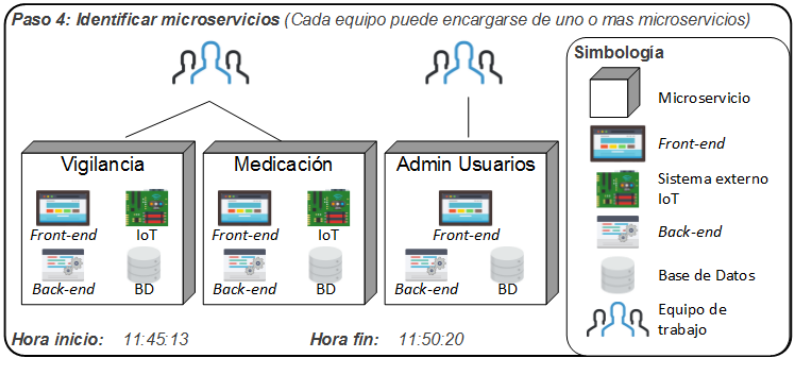


Figura 5: Identificación de microservicios para el enunciado de ejemplo.

Una vez identificados los microservicios se debe verificar si cumplen con los siguientes criterios:

- Cada microservicio tiene una única responsabilidad y se conforma de front-end back-end y su lógica de negocio.
- No hay llamadas que generen una alta conexión y dependencia entre microservicios
- Cada microservicio es lo suficientemente pequeño como para que un equipo pequeño lo pueda generar trabajando de forma independiente.
- Los servicios no están estrechamente acoplados y pueden evolucionar independientemente.
- Los límites de servicio no causarán problemas con la coherencia de datos o la integridad.
- Todas las entidades aparecen exactamente una vez en el esquema. Otras entidades pueden contener referencias a él, pero no lo duplican.

e) **Una vez validados los microservicios han sido validados con los criterios anteriores asignamos los equipos de trabajo y aplicamos criterios de modelo de datos.** Como se explica en la metodología MicroIoT (Anexo 1). Un mismo equipo de trabajo puede trabajar sobre uno o más microservicios, pero cada uno de los microservicios debe tener un equipo de trabajo a cargo del desarrollo y mantenimiento del mismo. Si un microservicio se comunicará con un dispositivo IoT, es necesario que cuente con un sistema externoIoT. Todo microservicio debe tener un back-end que representa la lógica que contiene el microservicio. Para el acceso a datos existe un criterio indiscutible que es el acceso a datos, dos o más microservicios pueden compartir una base de datos, pero nunca acceder a los mismos datos.

2. **Actividad 2: Adecuación de la Arquitctura de la Aplicación.**

- Las vistas de la aplicación hacen referencia a las interfaces gráficas o vistas de la aplicación para los diferentes tipos de usuarios.
- Los microservicios son simplemente las aplicaciones independientes que se han identificado, debe recordar que un microservicio tiene su propio frontend y este puede ser parte de n vistas.
- Los controladores de dispositivos son aquellos especificados en los requerimientos de hardware o los disponibles para la construcción de

la solución. Se recuerda que algunos dispositivos como los celulares o smartwatch son controladores y pueden tener integrados varios sensores o controlar otros externos.

- Los dispositivos de hardware son los sensores, camaras, focos, electrodomesticos, etc. Se consideran tanto sensores y actores.

En la figura 6 se muestra la adecuación de la arquitectura resultante para el enunciado del ejemplo, planteado en la actividad 1.

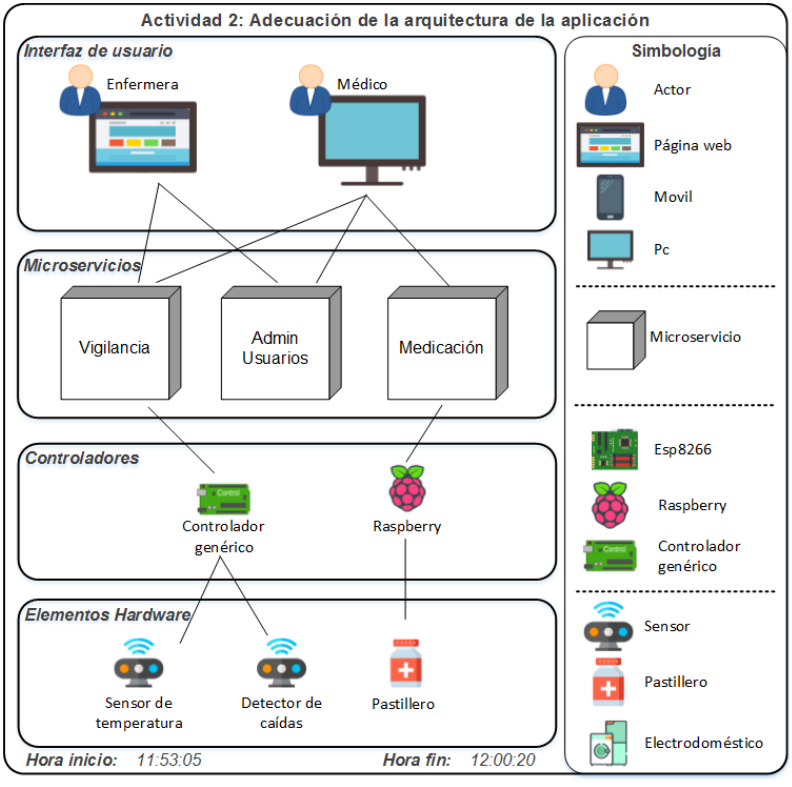


Figura6:Adecuación de la Arquitectura de la Aplicación para el enunciado de ejemplo.

A.7. Anexo de guía del cuasi-experimento

A.7.1. Anexo 1: Descripción de la Metodología

El desarrollo tradicional de aplicaciones siempre ha seguido un enfoque monolítico si bien se dividen funcionalidades en métodos y clases, toda la lógica de negocio está altamente acoplada dado que un cambio en la aplicación monolítica implica que se reconstruya y despliegue todo el sistema monolítico, al igual que el uso de recursos aumenta para toda la aplicación monolítica a medida que cualquier funcionalidad de la aplicación lo requiere.

A diferencia de las aplicaciones monolíticas, el mantenimiento en aplicaciones basadas en arquitectura de microservicios es más fácil debido a que únicamente se reconstruye el microservicio y no toda la aplicación, además de ello; este estilo de aplicaciones puede escalar con mayor facilidad a través del tiempo.

La metodología MicroIoT permite la creación de aplicaciones de Internet de las cosas desplegados en entornos de vida asistidos por el ambiente tales como hospitales u hogares, para ello se implementa la arquitectura basada en microservicios. Aprovechando los beneficios de los microservicios, la aplicación de esta metodología permite construir aplicaciones que puedan ser mantenibles con mayor facilidad, basándose en el principio de responsabilidad única que posee cada microservicio.Las actividades que comprende la metodología se presentan en la Figura 1.

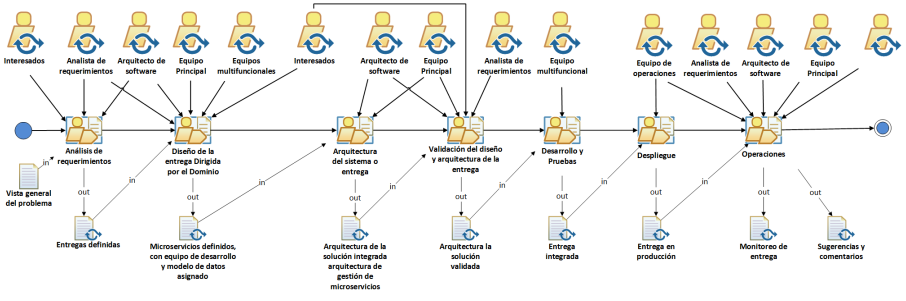


Figura 1: Flujo de procesos de Metodología MicroIoT.

1. **Análisis de requerimientos:**

   El análisis de requerimientos es vital para la planificación del desarrollo de una solución o sistema de software, dado que la metodología propuesta se centra en un desarrollo ágil se consideran iteraciones cíclicas para obtener entregas continuas a corto tiempo y reducir el tiempo de

puesta en producción de un producto en el mercado o una entrega que abarca requerimientos priorizados. Esta actividad comprende las tareas presentadas en la Figura 2.

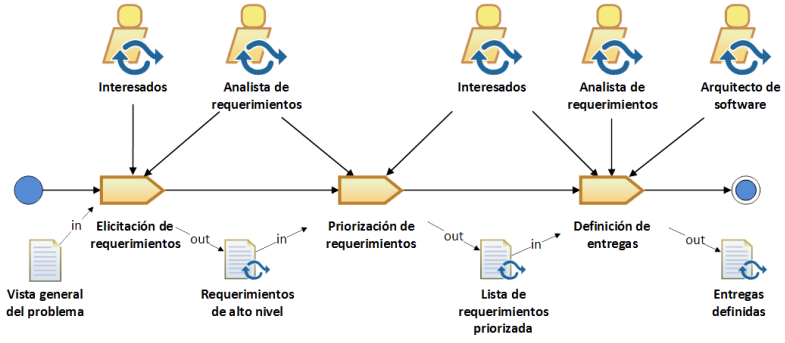


Figura 2: Tareas de actividad: "Análisis de Requerimientos".

1.1 **Elicitación de requerimientos.**

La elicitación de requerimientos tiene como objetivo obtener una vista general del problema o solución de IoT para AAL identificando: i) las funcionalidades generales y ii) entregables requeridos, a modo de requerimientos de alto nivel.

2.2 **Priorización de requerimientos**

La priorización es un paso crucial para tomar buenas decisiones con respecto a la planificación de productos para entregas únicas y múltiples. Las decisiones de priorización las toman los interesados, incluidos los usuarios, los gerentes, los desarrolladores o sus representantes como lo es el analista de requerimientos.

3.3 **Definición de entregas**

Las entregas de software son definidas a partir de las prioridades definidas en la tarea anterior. La primera entrega debe ser un poco más grande que las posteriores, pues esta debe servir de base para todas las demás entregas. Las entregas posteriores pueden ser sistemas independientes o simplemente incremento de funcionalidades en el sistema base de la primera entrega. Una entrega debe ser terminada en un plazo de 2 a 4 semanas, lo cual es característico de una metodología ágil.

2. **Diseño de la entrega Dirigida por el Dominio:**

Esta actividad comprende todas las actividades requeridas para obtener el conjunto de microservicios que conformarán la aplicación a partir de un conjunto de requerimientos proporcionados en la actividad anterior, los pasos comprendidos por esta actividad se presentan en la Figura 3.

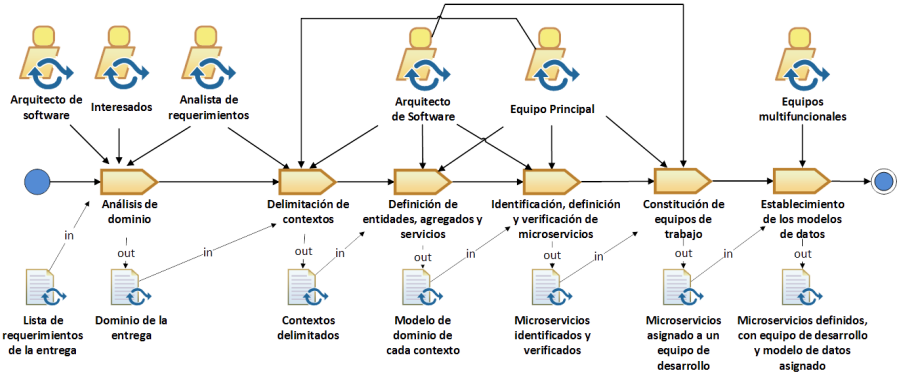


Figura 3: Actividades del Diseño de la entrega Dirigida por el Dominio.

2.1 Análisis de dominio

El análisis de dominio involucra identificar del problema, la solución que se pretende construir y sus requerimientos; es decir, efectuar iterativamente la elicitación de requerimientos recomendada en la actividad anterior.

El Analista de Requerimientos junto con el Arquitecto de Software del proyecto deben analizar el dominio en el que se desarrollan los requerimientos, identificando los controladores del negocio, el contexto del sistema, y los factores que el dueño del producto *(product owner)* y el experto de dominio, estimen crítico para el éxito.

2.2 Delimitación de contexto

Esta fase está a cargo del Analista de Requerimientos y el Arquitecto de Software. Un contexto delimitado es simplemente el límite dentro de un dominio, en donde se aplica un modelo de dominio en particular. En caso de existir contextos previamente identificados (de una entrega o iteración anterior), el Arquitecto de Software deberá hacer un mapeo para verificar si los nuevos cambios afectan o no a los contextos pre-existentes.

2.3 Definir entidades, agregados y servicios

Dentro de un contexto delimitado, se definen los modelos de dominio de cada contexto, para ello; aplicando modelado por dominio se define

las entidades, objetos de valor, agregados y servicios de dominio. A continuación se presentan algunas sugerencias para identificarlos.

- Una entidad, es un tipo de objeto requerido en la aplicación y que posee una identificación (por ejemplo: las personas poseen la cédula de identidad como identificación).
- Un objeto de valor es un elemento que carece de identidad.
- Un agregado hace referencia al tipo de objeto a partir del cual se derivan otras entidades(entidades padre o entidades que tienen componentes).
- Descomponer la aplicación por sustantivo ayuda a identificar las entidades y los objetos de valor involucrados en cada contexto.
- Usar la descomposición basada en verbos para definir los servicios existentes, y asignarlos a la entidad, agregado o tipo de valor, pertinente.

Las relaciones entre entidades, agregados y objetos de valor conforman el modelo de dominio perteneciente al microservicio.

2.4 **Identificación, definición y verificación de microservicios:**

2.4.1 **Identificación de microservicios.**
Un microservicio es aquel que cumple con todos los servicios ofrecidos en un determinado contexto. Los microservicios serán definidos a partir de los resultados obtenidos en la tareas anteriores de esta actividad. Como principio general, un microservicio no debe ser menor que un agregado ni mayor que un contexto delimitado, además se debe tener en cuenta que los agregados suelen ser buenos candidatos para microservicios.

2.4.2 **Definición de microservicios.**
Es esta fase se definen a continuación los componentes que regularmente abarca un microservicio cuando se pretende implementar sistemas de IoT para AAL basados en microservicios.

- ***Front-end:*** Este componente hace referencia las interfaces gráficas de usuario, están pueden ser web, móviles, o de escritorio.
- **Base de datos:** Cada microservicio abarca un subdominio del sistema de IoT para AAL al que pertenece, por ello debe manejar sus datos independientemente, (aunque comparta la base de datos, usa tablas exclusivas para ese microservicio), en las tareas posteriores se definirá un modelo de datos para cada microservicio.

- ***Back-end:*** Este componente hace referencia a la lógica de negocio, su implementación a nivel de código está basado el submodelo de dominio correspondiente, se encarga de gestion base de datos, cálculos, entre otros. Por lo general cuando se codifica, está dispuesto en capas: capa de dominio (Dom), capa de acceso a datos (Dao), y capa de servicios (Serv).
- **Sistemas externos de IoT:** Estos son sistemas que consumen o proveen servicios funcionales al microservicio. Estos están compuestos por controladores y elementos de hardware que pueden ser sensores o actuadores, también se consideran dispositivos que tienen controladores y sensores integrados.

2.4.3 Verificación de microservicios.

Una vez que el software se ha diseñado y se identificaron los microservicios, es importante que el arquitecto de software revise el modelo para verificar la correcta identificación de los microservicios. Para esta fase se revisa que los microservicios cumplan con las siguientes condiciones.

- Cada microservicio tiene una única responsabilidad y se conforma de frontend backend y su lógica de negocio.
- No hay llamadas que generen una alta conexión y dependencia entre microservicios. Si dividir la funcionalidad en dos microservicios hace que estos generen demasiada conversación, esto puede ser una indicación de que estas funciones deben estar en el mismo microservicio.
- Cada microservicio es lo suficientemente pequeño como para que un equipo pequeño lo pueda generar trabajando de forma independiente.
- No hay interdependencias que requieran la implementación de dos o más microservicios en sincronía.
- Los límites del microservicio no causarán problemas con la coherencia de datos o la integridad.
- Todas las entidades aparecen exactamente una vez en el esquema. Otras entidades pueden contener referencias a él, pero no lo duplican.

2.5 Constitución de equipos de trabajo

Tomando en cuenta la Ley de Conway *"Cualquier organización que diseña un sistema producirá un diseño cuya estructura es una copia*

de la estructura de comunicación de la organización", a cada microservicio se le asigna un equipo de desarrollo cuyos integrantes se especializan en diferentes áreas (equipos lógicos de bases de datos, equipos lógicos de lado de servidor, equipo de desarrollo de base de datos).

Dependiendo del número de integrantes del grupo de desarrollo, los equipos de trabajo pueden encargarse de varios microservicios.

2.6 **Establecimiento de modelos de datos**

Cada microservicio administra sus propios datos, la integridad y la coherencia de los datos son desafíos fundamentales. Un principio básico de los microservicios es que cada servicio administra sus propios datos. Dos servicios no deben compartir un mismo almacén de datos. En su lugar, cada servicio es responsable de su propio almacén de datos privado, al que otros servicios no pueden acceder directamente, se pueden compartir las bases de datos, pero nunca los mismos datos pueden ser usados por dos o más microservicios, como se muestra en la Figura 4.

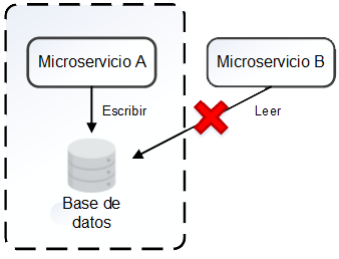


Figura 4: Consideraciones de bases de datos compartidas.

3. Arquitectura del sistema o entrega

3.1 Arquitectura del sistema o entrega

Con los microservicios previamente identificados, en esta actividad, la solución planteada se adapta a una Arquitectura general de IoT basada en microservicios. Esta arquitectura se presenta en la Figura 5.

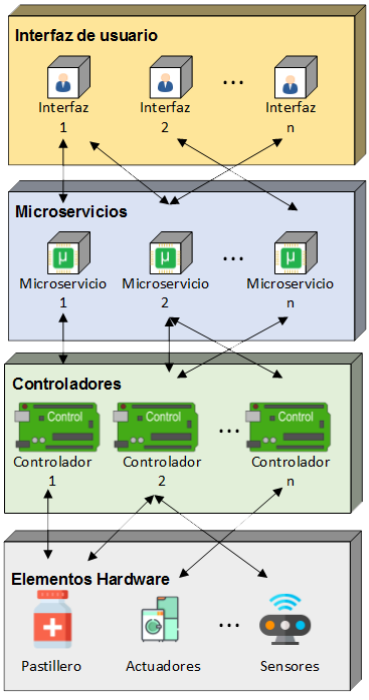


Figura 5: Arquitectura de la aplicación

- **Interfaz de usuario:** La interfaz de usuario se conforma con el grupo de front-ends proporcionados por los microservicios.
- **Capa de Microservicios:** Contiene los microservicios establecidos en el paso anterior.
- **Controladores:** Los controladores se encargan de comunicar los servicios proporcionados por los microservicios con los dispositivos de hardware, en caso
- **Elementos Hardware:** Los elementos de hardware constituyen todo elemento que sea de tipo sensor o actuador.

3.2 Establecimiento de arquitectura de gestión de microservicios

La arquitectura de gestión de microservicios corresponde a la adopción de un conjunto de patrones de microservicios que facilitan la gestión de diferentes tareas relacionadas con los microservicios. Al tener los esquemas de la estructura de los diferentes microservicios,

durante esta actividad el equipo de desarrollo determina qué patrones son los más adecuados para la aplicación. Entre los aspectos más relevantes tenemos:

- **Gestión de datos:** Al existir una independencia entre los diferentes microservicios, es necesario adoptar un estilo de gestión de datos que permita mantener la consistencia entre ellos, así como se requiere un estilo de consulta dependiendo de las necesidades de cada microservicio.
- **Descubrimiento de microservicios:** Corresponde a la estrategia a adoptar para determinar los microservicios que se están ejecutando junto con su ubicación (la ubicación corresponde a la dirección IP en la que está desplegada, y el puerto sobre el cual se está ejecutando).
- **El estilo de acceso a los microservicios:** Corresponde al tipo de consultas que se realizan para acceder a los servicios que publica los microservicios, las peticiones REST constituye el estilo de acceso más utilizado.
- **Balanceamiento de carga:** Este patrón perteneciente a cloud computing permite distribuir el trabajo entre un conjunto de servicios iguales.
- **Despliegue de microservicios:** Al adoptar un patrón de este tipo, se determina cómo se envían los microservicios a un entorno de producción. El tipo de despliegue de microservicios más utilizado es de contenedores.

4. **Desarrollo y pruebas**

   Esta actividad implica tres prácticas de desarrollo de software que se presentan como tareas de esta actividad: i) desarrollo continuo, ii) evaluación continua, y iii) integración continua.

   4.1 **Desarrollo continuo**

   En esta actividad cada equipo de trabajo se encarga de implementar o modificar un microservicio designado para que cumpla con determinadas funcionalidades requeridas. Para esta actividad se emplean herramientas que faciliten la gestión y el versionamiento de código.

   Luego de contar con el código desarrollado se requiere generar archivos ejecutables es por ello que se requiere herramientas que faciliten la gestión de dependencias y la compilación del código.

   4.2 **Evaluación continua**

En esta actividad el evaluador y el equipo de aseguramiento de calidad elaboran las pruebas necesarias para comprobar que los servicios del microservicio funcionan correctamente.

4.3 **Integración continua (CI)**

La integración continua corresponde a la compilación automatizada (incluidas las pruebas de calidad) para detectar errores de integración lo más rápido posible. Este enfoque conduce a problemas de integración significativamente reducidos y permite a un equipo desarrollar software cohesivo más rápidamente. En esta fase se realiza las siguientes actividades:

- Comprobación del código en el repositorio de código.
- Ejecución de pruebas unitarias.
- Las nuevas funcionalidades se integran con las preexistentes para buscar algún problema de integración.

5. **Despliegue**

El objetivo perseguido en esta fase es permitir lanzar nuevas funcionalidades a los clientes y usuarios lo más pronto posible reduciendo la intervención humana. Durante esta fase se debe considerar dos tipos de herramientas involucradas al entorno de despliegue:

- Herramientas que permitan gestionar el entorno, realizar y superar pruebas relacionadas al entorno (como pueden ser pruebas de rendimiento, funcionalidad o seguridad)
- Herramientas para la gestionar la configuración de aplicaciones en un entorno de despliegue.

6. **Operaciones**

Esta actividad tiene dos prácticas, que permiten: i) monitorear cómo las aplicaciones lanzadas se están desempeñando en la producción y ii) recibir retroalimentación de los clientes. Esta información permite a las empresas reaccionar de manera ágil y cambiar sus planes comerciales según sea necesario.

6.1 **Monitoreo contínuo**

El monitoreo continuo proporciona datos y métricas a las operaciones, al control de calidad, al personal de desarrollo de líneas de negocio y a otras partes interesadas sobre las aplicaciones en las diferentes etapas del ciclo de entrega, el monitoreo es una tarea automatizada mediante herramientas.

- **Herramientas de monitoreo:** Permiten que las organizaciones identifiquen y resuelvan los problemas de la infraestructura de TI (Tecnologías de la Información) antes de que afecten a los procesos comerciales críticos. Estas herramientas supervisan aspectos del sistema tales como la carga de la CPU, la asignación de RAM, las estadísticas de tráfico de red, el consumo de memoria y la disponibilidad de espacio libre en el disco.

6.2 Retroalimentación de los Interesados

Al estar en un entorno de producción, el sistema de software puede ser usado por los interesados y usuarios finales, en caso de surgir peticiones de cambios o nuevos requerimientos por parte de los interesados, también se deberá incurrir en una nueva iteración de la metodología, pero con un enfoque especial en la priorización de requerimientos para definir adecuadamente las entregas pendientes y el orden en el que serán puestas en producción para uso de los interesados y usuarios finales.

A.8. Ejemplo guía del cuasi-experimento

METODOLOGÍA Micro IoT

Nombres: ..... Apellidos: .....
Fecha: ..... Profesión: .....

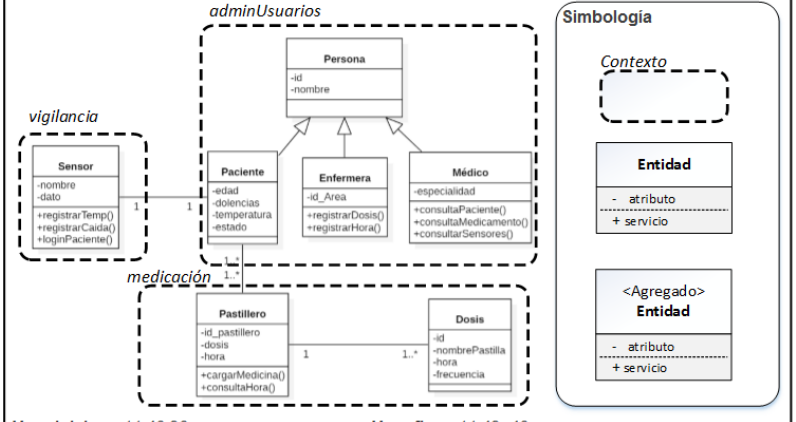
Descripción del problema: .
Un hospital desea automatizar muchas de sus tareas para proporcionar un servicio de vigilancia y medicación controlada a sus pacientes, para la vigilancia cada paciente contará con un controlador genérico que controla un sensor de temperatura corporal y un detector de caídas, las enfermeras del hospital deben tener acceso a una página web que les permita ver la información del paciente y la información de los 2 sensores mencionados anteriormente. Para la medicación controlada se cuenta con un raspberry que envía la información recolectada por el pastillero de cada paciente (dosis, horario) a la nube. Solamente los médicos podrán tener acceso a la información del pastillero, es decir; si se ha cumplido con el horario de medicación y cuál fue la dosis administrada. Además el médico también puede consultar la temperatura del paciente y el registro del detector de caídas a través de una aplicación de escritorio.

Actividad 1: Diseño de la Aplicación

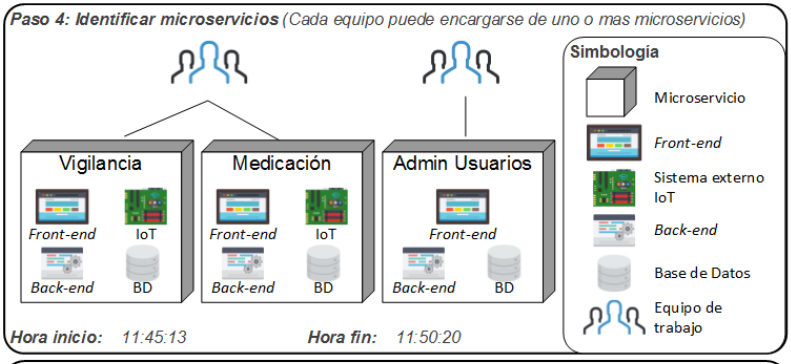
Paso 1: Análisis del dominio *(Identificación de requerimientos)*

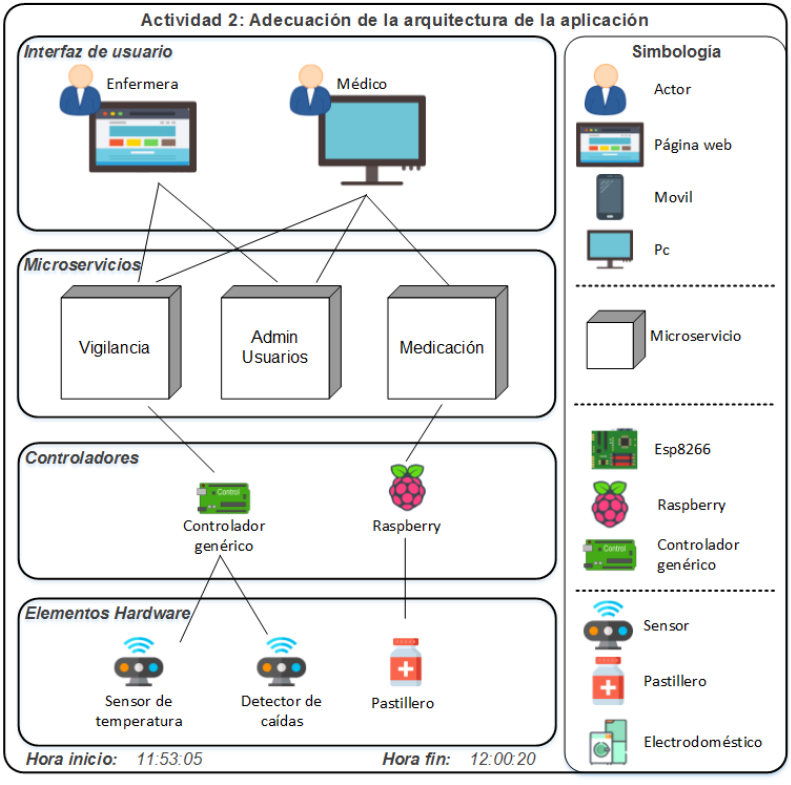
- Servicio de vigilancia cuenta con:
  - Controlador genérico se conecta con:
    - Sensor de temperatura corporal.
    - Sensor detector de caídas.
- Vista (Pagina Web) para enfermeras
  - Visualiza información del paciente.
  - Visualiza información de sensores.
- Servicio de monitoreo cuenta con:
  - Controlador Raspberry pi se conecta con:
    - Un pastillero que registra la dosis y el horario del medicamento.
- Vista (Aplicación de escritorio) para los médicos
  - Visualiza información del pastillero.
  - Visualiza información del paciente.
  - Visualiza información de los sensores.

Paso 2: Identifique contextos *(Cada contexto contiene un submodelo de dominio con entidades asociadas.)*



Hora inicio: *11:40:20* ***Hora fin:*** *11:42: 40*


Paso 4: Identificar microservicios (Cada equipo puede encargarse de uno o mas microservicios)
Vigilancia
Front-end
IoT
Back-end
BD
Medicación
Front-end
IoT
Back-end
BD
Admin Usuarios
Front-end
Back-end
BD
Hora inicio: 11:45:13
Hora fin: 11:50:20
Simbología
Microservicio
Front-end
Sistema externo IoT
Back-end
Base de Datos
Equipo de trabajo


Actividad 2: Adecuación de la arquitectura de la aplicación
Interfaz de usuario
Enfermera
Médico
Microservicios
Vigilancia
Admin Usuarios
Medicación
Controladores
Controlador genérico
Raspberry
Elementos Hardware
Sensor de temperatura
Detector de caídas
Pastillero
Hora inicio: 11:53:05
Hora fin: 12:00:20
Simbología
Actor
Página web
Movil
Pc
Microservicio
Esp8266
Raspberry
Controlador genérico
Sensor
Pastillero
Electrodoméstico

A.9. Ejercicio propuesto del cuasi-experimento

METODOLOGÍA Micro IoT

Nombres: ..... Apellidos: .....
Fecha: ..... Profesión: .....

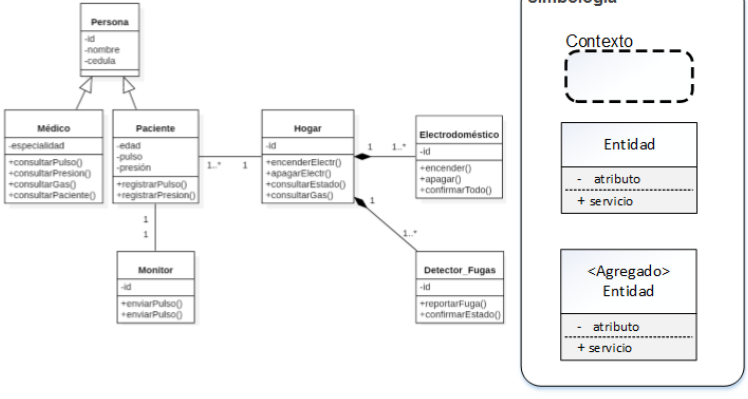
Descripción del problema: .
Un hospital desea implementar un sistema de monitoreo y asistencia en el hogar para sus pacientes (adultos mayores), para monitorear a los pacientes se cuenta con un controlador Raspberry pi que se conecta a un sensor de pulso y a otro sensor de presión arterial, los médicos podrán acceder a una página web para conocer los datos personales de su paciente y los signos vitales del mismo. Para brindar asistencia en el hogar del paciente se plantea una plataforma web en la que el usuario puede acceder para encender/apagar o conocer el estado de sus electrodomésticos inteligentes (sensores y controladores integrados) y observar los reportes de un detector de fugas de gas conectado a un ESP8266, por último, el médico puede observar también los reportes del detector de fugas de gas.

Actividad 1: Diseño de la Aplicación

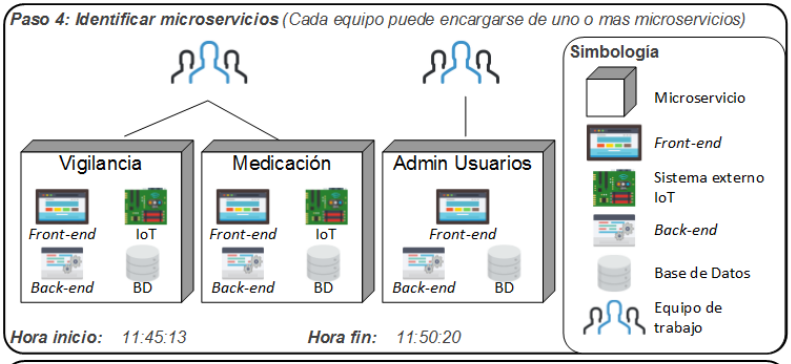
Paso 1: Análisis del dominio *(Identificación de requerimientos)*

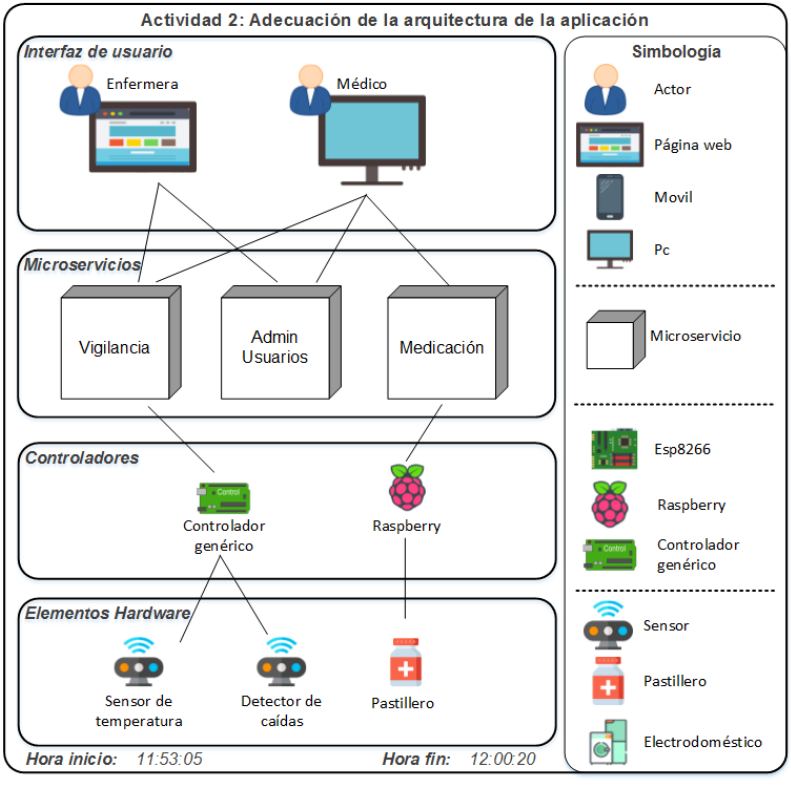
- Servicio de Monitoreo cuenta con:
  - Raspberry Pi se conecta con:
    - Sensor de pulso.
    - Sensor de presión arterial.
- Vista (Pagina Web) para Profesionales de la salud
  - Acceso a datos personales del paciente.
  - Consultar fuga de gas.
  - Visualiza información de sensores.
- Servicio de asistencia cuenta con:
  - Controlador genérico se conecta con:
    - Electrodomésticos.
  - Controlador ESP8266 se conecta con:
    - Detector de fugas de gas
- Vista (Página Web) para pacientes
  - Encender/apagar/consultar electrodomésticos.
  - Consultar fuga de gas.

Paso 2: Identifique contextos *(Cada contexto contiene un submodelo de dominio con entidades asociadas.)*
Paso 3: Identifique agregados. *(Los agregados hacen referencia a aquellas entidades raíz, de las cuales otras entidades se derivan, entidades padre o entidades que tienen componentes).*



Hora inicio: ***Hora fin:***


Paso 4: Identificar microservicios (Cada equipo puede encargarse de uno o mas microservicios)
Vigilancia
Front-end
IoT
Back-end
BD
Medicación
Front-end
IoT
Back-end
BD
Admin Usuarios
Front-end
Back-end
BD
Hora inicio: 11:45:13
Hora fin: 11:50:20
Simbología
Microservicio
Front-end
Sistema externo IoT
Back-end
Base de Datos
Equipo de trabajo


Actividad 2: Adecuación de la arquitectura de la aplicación
Interfaz de usuario
Enfermera
Médico
Microservicios
Vigilancia
Admin Usuarios
Medicación
Controladores
Controlador genérico
Raspberry
Elementos Hardware
Sensor de temperatura
Detector de caídas
Pastillero
Hora inicio: 11:53:05
Hora fin: 12:00:20
Simbología
Actor
Página web
Movil
Pc
Microservicio
Esp8266
Raspberry
Controlador genérico
Sensor
Pastillero
Electrodoméstico

A.10. Encuesta aplicada a cuasi-experimento

Evaluación de metodología "MicroIoT" enfocada en aplicaciones basadas en Microservicios para soluciones de Internet de las Cosas (IoT) en Ambientes de Vida Asistida (AAL).

Para cada una de las preguntas marque, por favor, una cruz sobre el círculo que se encuentra lo más cerca posible de su opinión.
LEA POR FAVOR CADA PREGUNTA CUIDADOSAMENTE ANTES DE DAR SU RESPUESTA

*Obligatorio

1. La forma de aplicar la metodología me ha parecido compleja y difícil de seguir *
Marca solo un óvalo.

	1	2	3	4	5	
Totalmente de Acuerdo	◯	◯	◯	◯	◯	Totalmente en Desacuerdo

2. Creo que la metodología reduciría el tiempo y el esfuerzo requerido para crear sistemas basados en microservicios soluciones de IoT en AAL *
Marca solo un óvalo.

	1	2	3	4	5	
Totalmente en Desacuerdo	◯	◯	◯	◯	◯	Totalmente de Acuerdo

3. De manera general, la metodología "MicroIoT" es difícil de entender *
Marca solo un óvalo.

	1	2	3	4	5	
Totalmente de Acuerdo	◯	◯	◯	◯	◯	Totalmente en Desacuerdo

4. Los pasos propuestos por la metodología para diseñar una aplicación son claros y fáciles de entender *
Marca solo un óvalo.

	1	2	3	4	5	
Totalmente en Desacuerdo	◯	◯	◯	◯	◯	Totalmente de Acuerdo

5. De manera general, considero que la metodología es útil *
Marca solo un óvalo.

	1	2	3	4	5	
Totalmente en Desacuerdo	◯	◯	◯	◯	◯	Totalmente de Acuerdo

6. La metodología es difícil de aprender *

Marca solo un óvalo.

	1	2	3	4	5	
Totalmente de Acuerdo	◯	◯	◯	◯	◯	Totalmente en Desacuerdo

7. Creo que la actividad 1: "Diseño de la aplicación" ayuda a identificar los microservicios adecuados para la creación de sistemas basados en microservicios para soluciones de IoT en AAL *

Marca solo un óvalo.

	1	2	3	4	5	
Totalmente en Desacuerdo	◯	◯	◯	◯	◯	Totalmente de Acuerdo

8. Creo que la actividad 2: "Adecuación de la arquitectura de la aplicación" ayuda a identificar los todos los componentes requeridos para la creación de sistemas basados en microservicios para soluciones de IoT en AAL *

Marca solo un óvalo.

	1	2	3	4	5	
Totalmente en Desacuerdo	◯	◯	◯	◯	◯	Totalmente en Acuerdo

9. Si tuviera que utilizar una metodología para aplicaciones basadas en microservicios para soluciones de IoT en AAL en el futuro, creo que tendría en cuenta esta metodología *

Marca solo un óvalo.

	1	2	3	4	5	
Totalmente en Desacuerdo	◯	◯	◯	◯	◯	Totalmente de Acuerdo

10. Creo que la metodología NO es lo suficientemente expresiva para definir como los componentes interactúan entre sí *

Marca solo un óvalo.

	1	2	3	4	5	
Totalmente de Acuerdo	◯	◯	◯	◯	◯	Totalmente en Desacuerdo

11. El uso de esta metodología mejoraría mi rendimiento en la creación de sistemas basados en microservicios para soluciones de IoT en AAL *

Marca solo un óvalo.

	1	2	3	4	5	
Totalmente en Desacuerdo	◯	◯	◯	◯	◯	Totalmente de Acuerdo

12. Pienso que sería fácil ser hábil usando esta metodología *

Marca solo un óvalo.

	1	2	3	4	5	
Totalmente en Desacuerdo	◯	◯	◯	◯	◯	Totalmente de Acuerdo

13. De manera general, pienso que esta metodología NO puede cubrir adecuadamente los requisitos para sistemas basados en microservicios para soluciones de IoT en AAL *
Marca solo un óvalo.

	1	2	3	4	5	
Totalmente de Acuerdo	◯	◯	◯	◯	◯	Totalmente en Desacuerdo

14. En caso de tener la necesidad de crear sistemas basados en microservicios para soluciones de IoT en AAL, tendría la intención de utilizar está metodología en el futuro *
Marca solo un óvalo.

	1	2	3	4	5	
Totalmente en Desacuerdo	◯	◯	◯	◯	◯	Totalmente de Acuerdo

15. No recomendaría el uso de esta metodología *
Marca solo un óvalo.

	1	2	3	4	5	
Totalmente de Acuerdo	◯	◯	◯	◯	◯	Totalmente en Desacuerdo

16. ¿Tiene alguna sugerencia de cómo hacer que esta metodología sea más fácil de usar? *

17. ¿Cuáles son las razones por las que tiene o no la intención de usar esta metodología en un futuro? *

18. **Por favor ingrese su nombre** *

A.11. Ejercicio propuesto del cuasi-experimento (Resuelto)

METODOLOGÍA Micro IoT

Nombres: ..... Apellidos: .....
Fecha: ..... Profesión: .....

Descripción del problema: .

Un hospital desea implementar un sistema de monitoreo y asistencia en el hogar para sus pacientes (adultos mayores), para monitorear a los pacientes se cuenta con un controlador Raspberry pi que se conecta a un sensor de pulso y a otro sensor de presión arterial, los médicos podrán acceder a una página web para conocer los datos personales de su paciente y los signos vitales del mismo. Para brindar asistencia en el hogar del paciente se plantea una plataforma web en la que el usuario puede acceder para encender/apagar o conocer el estado de sus electrodomésticos inteligentes (sensores y controladores integrados) y observar los reportes de un detector de fugas de gas conectado a un ESP8266, por último, el médico puede observar también los reportes del detector de fugas de gas.

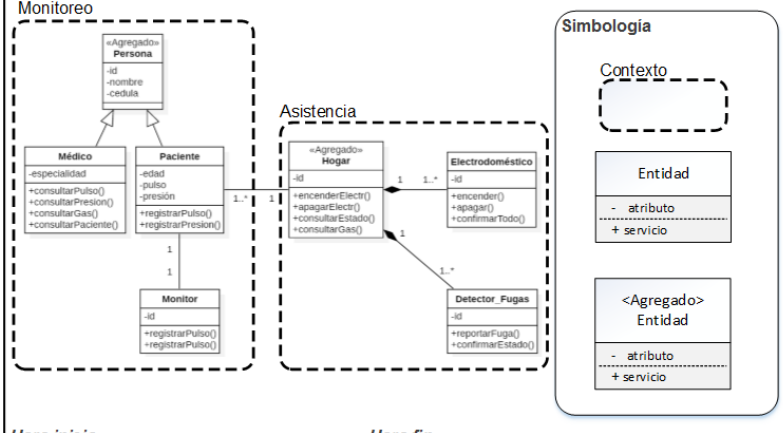
Actividad 1: Diseño de la Aplicación

Paso 1: Análisis del dominio *(Identificación de requerimientos)*

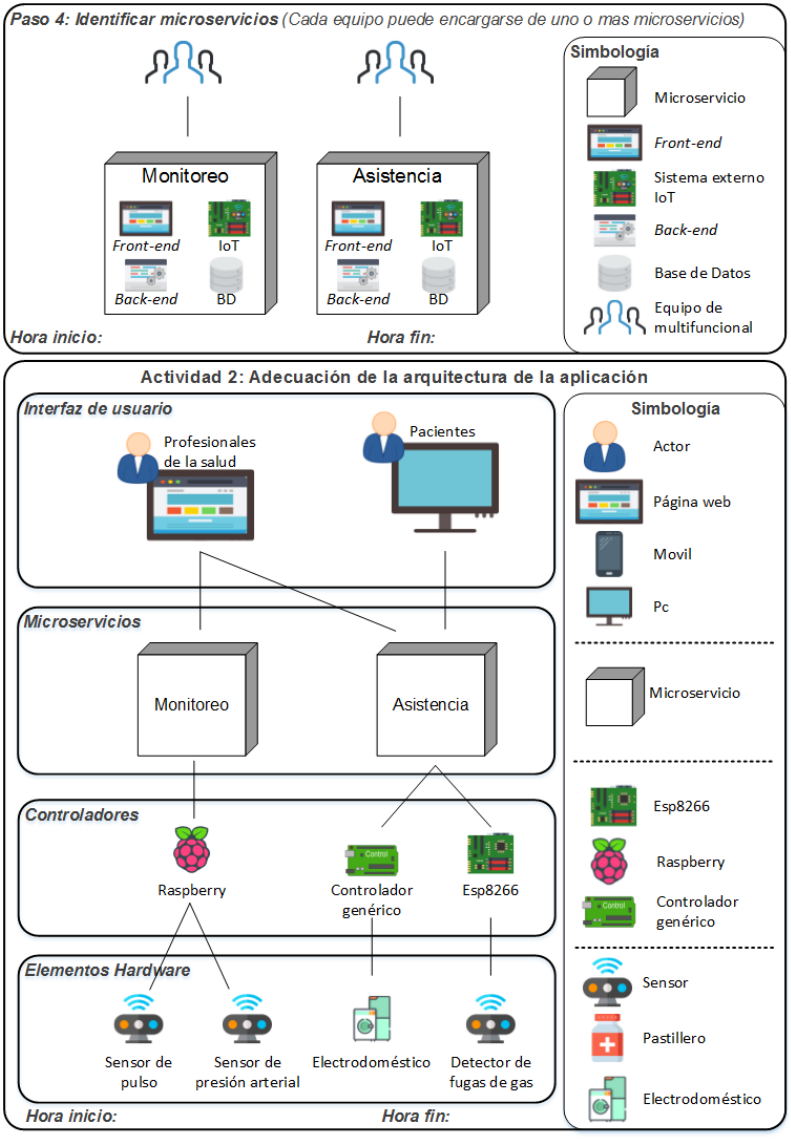
- Servicio de Monitoreo cuenta con:
  - Raspberry Pi se conecta con:
    - Sensor de pulso.
    - Sensor de presión arterial.
- Vista (Pagina Web) para Profesionales de la salud
  - Acceso a datos personales del paciente.
  - Consultar fuga de gas.
  - Visualiza información de sensores.
- Servicio de asistencia cuenta con:
  - Controlador genérico se conecta con:
    - Electrodomésticos.
  - Controlador ESP8266 se conecta con:
    - Detector de fugas de gas
- Vista (Página Web) para pacientes
  - Encender/apagar/consultar electrodomésticos.
  - Consultar fuga de gas.

Paso 2: Identifique contextos *(Cada contexto contiene un submodelo de dominio con entidades asociadas.)*

Paso 3: Identifique agregados. *(Los agregados hacen referencia a aquellas entidades raíz, de las cuales otras entidades se derivan, entidades padre o entidades que tienen componentes).*



Hora inicio: ***Hora fin:***


Paso 4: Identificar microservicios (Cada equipo puede encargarse de uno o mas microservicios)
Monitoreo
Front-end
IoT
Back-end
BD
Asistencia
Front-end
IoT
Back-end
BD
Hora inicio:
Hora fin:
Simbología
Microservicio
Front-end
Sistema externo IoT
Back-end
Base de Datos
Equipo de multifuncional
Actividad 2: Adecuación de la arquitectura de la aplicación
Interfaz de usuario
Profesionales de la salud
Pacientes
Microservicios
Monitoreo
Asistencia
Controladores
Raspberry
Controlador genérico
Esp8266
Elementos Hardware
Sensor de pulso
Sensor de presión arterial
Electrodoméstico
Detector de fugas de gas
Hora inicio:
Hora fin:
Simbología
Actor
Página web
Movil
Pc
Microservicio
Esp8266
Raspberry
Controlador genérico
Sensor
Pastillero
Electrodoméstico

Referencias

Abrahão, S., Insfran, E., Carsí, J. A., and Genero, M. (2011). Evaluating requirements modeling methods based on user perceptions: A family of experiments. *Information Sciences*, 181(16):3356–3378.

Acevedo, C. A. J., y Jorge, J. P. G., and Patiño, I. R. (2017). Methodology to transform a monolithic software into a microservice architecture. In *Software Process Improvement (CIMPS), 2017 6th International Conference on*, pages 1–6. IEEE.

Ashton, K. et al. (2009). That 'internet of things' thing. *RFID journal*, 22(7):97–114.

Aurum, A. and Wohlin, C. (2005). Requirements engineering: setting the context. In *Engineering and managing software requirements*, pages 1–15. Springer.

Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2016). Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, 33(3):42–52.

Basili, V. R. (1993). The experimental paradigm in software engineering. In *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, pages 1–12. Springer.

Basili, V. R., Selby, R. W., and Hutchens, D. H. (1986). Experimentation in software engineering. *IEEE Transactions on software engineering*, (7):733–743.

Bell, J., Ajontech, L., Currier, B., Harrington, E. E., Helstrom, C. B., and Martins, M. (2016). Microservices architecture.

Beltrán, G. and Óscar, A. (2005). Revisiones sistemáticas de la literatura. *Revista Colombiana de Gastroenterología*, 20(1).

Benítez-Gutiérrez, G. (2017). Ciudad digital: paradigma de la globalización urbana. *Revista Bitácora Urbano Territorial*, 27(1).

Brown, K. (2016). Una breve historia de los patrones de microservicios. *IBM DeveloperWorks*.

Butzin, B., Golatowski, F., and Timmermann, D. (2016). Microservices ap-

proach for the internet of things. In *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*, pages 1–6. IEEE.

Canós, J. H. and Letelier, M. C. P. P. (2012). Metodologías ágiles en el desarrollo de software.

Castro, D., Coral, W., Cabra, J., Colorado, J., Méndez, D., and Trujillo, L. (2017). Survey on iot solutions applied to healthcare. *Dyna*, 84(203):192–200.

Cedillo, I. P. (2014). Un método de evaluación de usabilidad de mashups basado en la composicionalidad de sus componentes.

Cedillo, I. P. (2017). *Monitorización de calidad de servicios cloud mediante modelos en tiempo de ejecución*. PhD thesis.

Conway, M. E. (1968). How do committees invent. *Datamation*, 14(4):28–31.

Cook, T. D. and Campbell, D. T. (1979). *Quasi-experimentation: Design and analysis for field settings*, volume 3. Rand McNally Chicago.

Cortés, M. E. C. and León, M. I. (2005). *Generalidades sobre Metodología de la Investigación*. Universidad Autónoma del Carmen.

Davis, F. D. (1985). *A technology acceptance model for empirically testing new end-user information systems: Theory and results*. PhD thesis, Massachusetts Institute of Technology.

de la Torre, C. (2017). Diseño de un microservicio orientado a un ddd.

de Lorenzo García, R. (2012). *El futuro de las personas con discapacidad en el mundo. Desarrollo humano y discapacidad*. Fundación Once.

Duarte, A. O. and Rojas, M. (2008). Las metodologías de desarrollo ágil como una oportunidad para la ingeniería del software educativo. *Avances en Sistemas e Informática*, 5(2).

Duplaga, M. (2013). The acceptance of e-health solutions among patients with chronic respiratory conditions. *Telemedicine and e-Health*, 19(9):683–691.

Dyck, A., Penners, R., and Lichter, H. (2015). Towards definitions for release engineering and devops. In *Release Engineering (RELENG), 2015 IEEE/ACM 3rd International Workshop on*, pages 3–3. IEEE.

Ebert, C., Gallardo, G., Hernantes, J., and Serrano, N. (2016). Devops. *IEEE Software*, 33(3):94–100.

Evans, D. (2011). The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1(2011):1–11.

Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

Fenech, T. (1998). Using perceived ease of use and perceived usefulness to predict acceptance of the world wide web. *Computer Networks and ISDN Systems*, 30(1-7):629–630.

Fernández, L. F. (2006). Arquitectura de software. *Software Guru*, 2(3):40–45.

Figueroa, R. G., Solís, C. J., and Cabrera, A. A. (2008). Metodologías tradicionales vs. metodologías ágiles. *Universidad Técnica Particular de Loja, Escuela de Ciencias de la Computación*.

Fishbein, M. and Ajzen, I. (1975). *Belief, attitude, intention and behavior: An introduction to theory and research*.

Fowler, M. and Foemmel, M. (2006). Continuous integration. *Thought-Works) http://www. thoughtworks. com/Continuous Integration. pdf*, 122:14.

Garriga, M. (2017). Towards a taxonomy of microservices architectures. In *International Conference on Software Engineering and Formal Methods*, pages 203–218. Springer.

Georgieff, P. (2008). Ambient assisted living. *Marktpotenziale IT-unterstützter Pflege für ein selbstbestimmtes Altern, FAZIT Forschungsbericht*, 17:9–10.

Gorschek, T., Garre, P., Larsson, S., and Wohlin, C. (2006). A model for technology transfer in practice. *IEEE software*, 23(6):88–95.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660.

Gutiérrez, L. (2010). Arquitectura software. *Investigación Aplicada a la Construcción de Marcos de Trabajo-Bucaramanga, Colombia:(Sic) Editorial Ltda*.

Hail, M. A. and Fischer, S. (2015). Iot for aal: An architecture via information-centric networking. In *Globecom Workshops (GC Wkshps), 2015 IEEE*, pages 1–6. IEEE.

Haywood, D. (2009). *Domain-driven design using naked objects*. Pragmatic Bookshelf.

Httermann, M. (2012). *DevOps for developers*. Apress.

Jaramillo, D. and Palacios, J. Elicitación de requisitos de resiliencia para sistemas de información basado en el modelo cert-rmm requeriments elicitaction of resilience for systems information based on the model cert-rmm.

Kampmeijer, R., Pavlova, M., Tambor, M., Golinowska, S., and Groot, W. (2016). The use of e-health and m-health tools in health promotion and primary prevention among older adults: a systematic literature review. *BMC health services research*, 16(5):290.

Kim, G., Debois, P., Willis, J., and Humble, J. (2016). *The DevOps handbook: how to create world-class agility, reliability, and security in technology organizations*. IT Revolution.

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., and Linkman, S. (2009). Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, 51(1):7–15.

Koreshoff, T. L., Robertson, T., and Leong, T. W. (2013). Internet of things: a review of literature and products. In *Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, pages 335–344. ACM.

Krylovskiy, A., Jahn, M., and Patti, E. (2015). Designing a smart city internet of things platform with microservice architecture. In *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, pages 25–30. IEEE.

Lewis, J. and Fowler, M. (2014). Microservices: a definition of this new architectural term. *Mars*.

Manifiesto, A. (2001). Principios de manifiesto ágil.

Molina, S. G. R. (2014). Metodologías ágiles enfocadas al modelado de requerimientos. *Informes Científicos-Técnicos UNPA*, 5(1):1–29.

Moody, D. L. (2003). The method evaluation model: a theoretical model for validating information systems design methods. *ECIS 2003 proceedings*, page 79.

Pahl, C. and Jamshidi, P. (2016). Microservices: A systematic mapping study. In *CLOSER (1)*, pages 137–146.

Patel, K. K., Patel, S. M., and Professor, P. S. A. (2016). Internet of things-iot: definition, characteristics, architecture, enabling technologies, application & future challenges. *Int. J. Eng. Sci. Comput*, 6(5).

Pieper, M., Antona, M., and Cortés, U. (2011). Ambient assisted living. *Ercim News*, 87:19.

Rademacher, F., Sachweh, S., and Zündorf, A. (2017). Differences between model-driven development of service-oriented and microservice architecture. In *Software Architecture Workshops (ICSAW), 2017 IEEE International Conference on*, pages 38–45. IEEE.

Rahimian, V. and Ramsin, R. (2008). Designing an agile methodology for mobile software development: A hybrid method engineering approach. In *Research Challenges in Information Science, 2008. RCIS 2008. Second International Conference on*, pages 337–342. IEEE.

Rescher, N. (1973). The primacy of practice.

Richardson, C. (2014). Pattern: Microservices architecture. *Microservices. io. http://microservices. io/patterns/microservices. html [last accessed on February 17, 2015]*.

Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131.

Schubert, P. and Dettling, W. (2002). Extended web assessment method (ewam)-evaluation of e-commerce applications from the customer's viewpoint. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual*

Hawaii International Conference on, pages 10–pp. IEEE.

Sethi, P. and Sarangi, S. R. (2017). Internet of things: architectures, protocols, and applications. *Journal of Electrical and Computer Engineering*, 2017.

Shadija, D., Rezai, M., and Hill, R. (2017). Towards an understanding of microservices. In *Automation and Computing (ICAC), 2017 23rd International Conference on*, pages 1–6. IEEE.

Sharma, S. (2017). *Mastering Microservices with Java 9: Build domain-driven microservice-based applications with Spring, Spring Cloud, and Angular*. Packt Publishing Ltd.

Sharma, S. and Coyne, B. (2013). Devops for dummies. *Limited IBM Edition'book*.

Sillitti, A. and Succi, G. (2005). Requirements engineering for agile methods. In *Engineering and Managing Software Requirements*, pages 309–326. Springer.

Steinegger, R. H., Giessler, P., Hippchen, B., and Abeck, S. (2017). Overview of a domain-driven design approach to build microservice-based applications. In *SOFTENG: The Third International Conference on Advances and Trends in Software Engineering*.

Sun, L., Li, Y., and Memon, R. A. (2017). An open iot framework based on microservices architecture. *China Communications*, 14(2):154–162.

Thönes, J. (2015). Microservices. *IEEE software*, 32(1):116–116.

Tinoco Gómez, O., Rosales López, P. P., and Salas Bacalla, J. (2010). Criterios de selección de metodologías de desarrollo de software. *Industrial Data*, 13(2).

Uviase, O. and Kotonya, G. (2018). Iot architectural framework: Connection and integration framework for iot systems. *arXiv preprint arXiv:1803.04780*.

Vavpotič, D., Bajec, M., and Krisper, M. (2004). Software development methodology evaluation model. In *Constructing the Infrastructure for the Knowledge Economy*, pages 141–153. Springer.

Vermesan, O. and Friess, P. (2014). *Internet of things-from research and innovation to market deployment*, volume 29. River publishers Aalborg.

Vernon, V. (2013). Implementing domain-driven design.

Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., and Gil, S. (2015). Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In *Computing Colombian Conference (10CCC), 2015 10th*, pages 583–590. IEEE.

Vresk, T. and Čavrak, I. (2016). Architecture of an interoperable iot platform based on microservices. In *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2016 39th International Convention on*, pages 1196–1201. IEEE.

Vural, H., Koyuncu, M., and Guney, S. (2017). A systematic literature review on microservices. In *International Conference on Computational Science and Its Applications*, pages 203–217. Springer.

Wasson, M. (2017). Análisis de dominios para microservicios. *Microsoft*.

Wiggins, A. (2011). The twelve-factor app. *The Twelve-Factor App*.

Wohlin, C. et al. (2005). *Engineering and managing software requirements*. Springer Science & Business Media.

Wolff, E. (2016). *Microservices: flexible software architecture*. Addison-Wesley Professional.

Yang, G., Xie, L., Mäntysalo, M., Zhou, X., Pang, Z., Da Xu, L., Kao-Walter, S., Chen, Q., and Zheng, L.-R. (2014). A health-iot platform based on the integration of intelligent packaging, unobtrusive bio-sensor, and intelligent medicine box. *IEEE transactions on industrial informatics*, 10(4):2180–2191.